CELEPAR
INFORMÁTICA
do PARANÁ

Desenvolvimento de Processos de Workflow utilizando o Expresso^{Livre}/Galaxia

Versão 2.3

Abril/2012

Sumário de Informações do Documento		
Tipo do Documento: Apostila		
Título do Documento: Desenvolvimento de Processos de Workflow Utilizando o ExpressoLivre/Galaxia		
Estado do Documento: Concluído		
Responsáveis: Maurício Luiz Viani, Adeildo Fernandes dos Santos		
Colaboradores: Sídnei Augusto Drovetto Junior, Carlos Eduardo Nogueira Gonçalves, Cassio Maes da Fonseca, Guilherme Striquer Bisotto		
Palavras-Chaves: expresso, workflow, galaxia, processos, atividades		
Resumo: esta apostila demonstra o uso do módulo workflow do ExpressoLivre, para desenvolvimento de fluxos de trabalho.		
Número de páginas: 95		
Software utilizados: ExpressoLivre, PostgreSQL, Eclipse		
Versão	**Data**	**Mudanças**
1.0	03 de agosto de 2007	Versão inicial
2.0	10 de julho de 2008	Atualização
2.1	03 de Setembro de 2008	Atualização
2.2	30 de julho de 2009	Atualização
2.3	3 de maio de 2012	Atualização

Equipe Técnica

Maurício Luiz Viani	GSR/DISER
Adeildo Fernandes dos Santos	GSR/DISER

Colaboradores

Sídnei Augusto Drovetto Junior

Carlos Eduardo Nogueira Gonçalves

Cássio Maes da Fonseca

Guilherme Striquer Bisotto

Sumário

Lista de Figuras

1 INTRODUÇÃO

O Expresso Livre é uma solução de correio eletrônico, agenda e catálogo de endereços inteiramente desenvolvida em software livre. Sendo uma versão customizada da ferramenta alemã E-Groupware[1], o seu objetivo é fornecer aos usuários do governo do estado uma ferramenta econômica e eficiente para suprir esta necessidade, já que outras ferramentas proprietárias, antes usadas, representavam alto custo para o Estado.

Além dos módulos básicos, citados anteriormente, o Expresso Livre conta com um módulo de workflow, cuja função principal é prover mecanismos para o desenvolvimento, controle e execução de fluxos de trabalho, conhecidos mais comumente como processos de workflow. Para alcançar este objetivo, o módulo de workflow conta com três interfaces: usuários, administração e monitoramento. Por trás destas interfaces existe um motor que foi incorporado ao módulo a partir de um projeto externo, o TikiWiki[2]. O motor de workflow, batizado de Galaxia[3], é o núcleo operacional do módulo, e as interfaces são apenas camadas, sobre este motor, para facilitar o acesso às suas funções.

[1] E-Groupware disponível em `http://www.egroupware.org`
[2] TikiWiki disponível em `http:/info.tiki.org`
[3] Galaxia disponível em `http://workflow.tikiwiki.org/tiki-index.php?page=GalaxiaConcepts`

2 CONCEITOS BÁSICOS

Esta seção é uma tradução/adaptação do documento Galaxia Introduction[4], originalmente produzido por Garland Foster, Richard Moore e Eduardo Polidor, e editado por Georger Araujo, para ser incorporado ao projeto TikiWiki. Marc Laport foi o primeiro integrante da equipe Tiki a sugerir a adição do motor de workflow ao Tiki.

2.1 Introdução

Galaxia é um workflow baseado em atividades. Os processos de workflow são implementados como um conjunto de atividades que devem ser completadas para atingir um resultado. As atividades do Galaxia são representadas por scripts PHP. São três os principais módulos do Galaxia: “administração de processos”, “controle de processos” e “interface do usuário”.

2.2 Definições

2.2.1 Processo

Um processo é definido como um conjunto de atividades que devem ser executadas para atingir um objetivo. Um processo no Galaxia equivale a um processo de negócio da empresa. As atividades do processo são conectadas entre si através de transições, definindo o que deve ser feito quando um atividade for completada.

2.2.2 Atividade

Uma atividade é algo que deve ser feito como parte de um processo. No Galaxia as atividades são mapeadas a scripts PHP (páginas), de modo que uma atividade pode fazer qualquer coisa possível de ser feita com PHP.

[4] *Galaxia Introduction* disponível em `http://prdownloads.sourceforge.net/tikiwiki/Galaxia_introduction.pdf?download`

2.2.3 Transição

As transições são as ligações entre as atividades. Elas definem o fluxo do processo, ou seja, a sequência em que as atividades devem ser executadas.

2.2.4 Perfil

Perfis definem quem pode executar uma atividade. Os perfis são criados a nível de processo e são associados à usuários e grupos.

2.2.5 Instância

Uma instância é um processo em execução. É criada quando um processo é iniciado e finalizada ao fim da sua atividade final.

2.2.6 Ítem de Trabalho

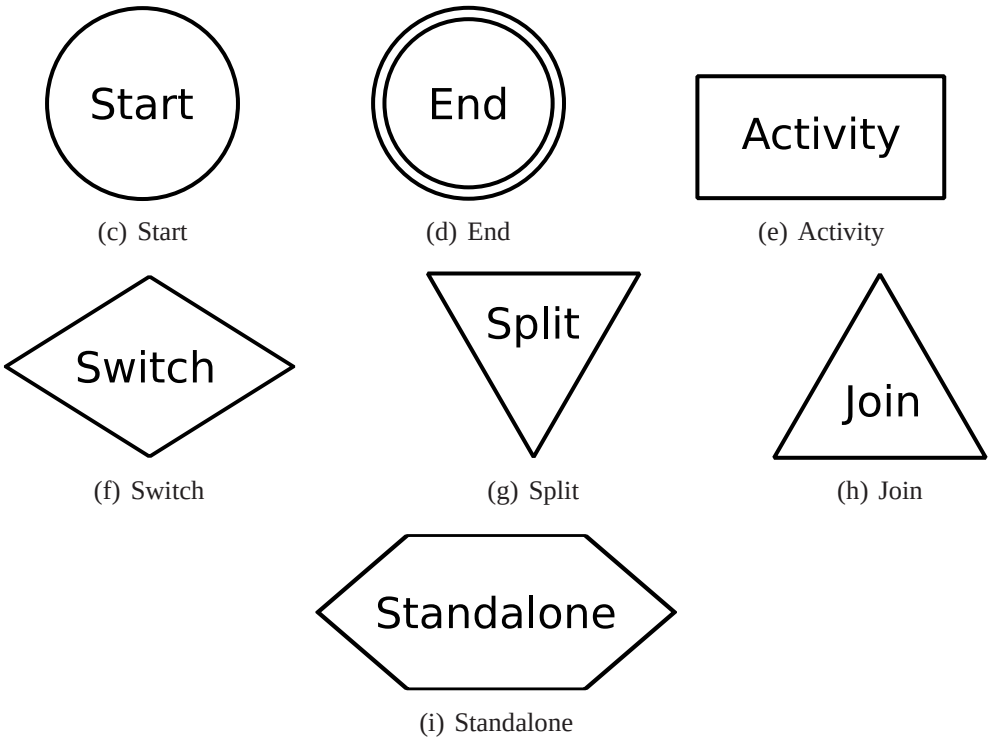
Quando uma atividade é completada, um ítem de trabalho é adicionado à instância e representam as atividades realizadas.

2.2.7 Job

Jobs são códigos vinculados a processos que podem ser executados em uma determinada data e hora sem a intervensão de qualquer usuário. Possuem uma finalidade específica e podem desempenhar deste papéis de monitoramento até comandar a execução de uma atividade.

2.3 Tipos de Atividades

Existem sete tipos de atividades que podem ser utilizadas para desenhar um processo. Estas atividades estão representadas na Figura 2.1 e são descritas mais adiante.



(c) Start (d) End (e) Activity

(f) Switch (g) Split (h) Join

(i) Standalone

Figura 2.1: Representação das atividades.

2.3.1 Start

Atividades do tipo **start** são representadas por um círculo. Ela é a responsável por criar a instância do processo. Tanto atividades deste tipo quanto as standalone são as únicas que podem ser executadas sem que o processo esteja instanciado. Um processo deve ter, pelo menos, uma atividade de start, porém podem existir mais de uma, apesar de isto não ser muito comum. Não podem existir transições de entrada para estas atividades, no entanto, deve existir pelo menos uma transição de saída.

2.3.2 End

No Galaxia a atividade **end** é representada usando um círculo duplo. A atividade end representa o fim do processo. Quando uma instância chega ao fim de uma atividade deste tipo, o processo é considerado concluído. Processos podem ter apenas uma atividade end. Isto não significa que um processo não possa terminar de diferentes maneiras, isto depende das atividades executadas antes desta atividade. Uma atividade end deve ter pelo menos uma transição

de entrada.

Regras: um processo para ser válido, deve ter ao menos um atividade start e apenas uma atividade end. Deve existir ao menos um caminho que leve do início ao fim.

2.3.3 Normal

Atividades do tipo **normal** são representadas por um retângulo. Elas não tem nenhum significado especial e são usadas para representar coisas que devam ser executadas como parte do processo. Atividades deste tipo podem receber diversas transições de entrada, mas apenas uma transição de saída.

2.3.4 Switch

Atividades do tipo **switch** são representadas por um losângulo. São atividades de decisão, isto é, são pontos onde o fluxo pode seguir caminhos diferentes, como cruzamentos de ruas. As instâncias que chegam a uma atividade deste tipo são avaliadas e dependendo do resultado a instância é desviada para um caminho ou outro. Estas podem ter muitas transições de entrada e muitas transições de saída.

2.3.5 Split

Atividades do tipo **split** são representadas por um triângulo em pé. Este tipo de atividade divide a instância em várias “subinstâncias” e estas poderão ser executadas em paralelo. Neste caso, é verdadeiro afirmar que uma instância pode estar em muitas atividades ao mesmo tempo. Este tipo de atividade pode receber diversas transições de entrada e pode ter diversas transições de saída.

2.3.6 Join

Atividades do tipo **join** são representadas por um triângulo invertido. Este tipo de atividade é usado para reagrupar instâncias que foram separadas por uma atividade split. Quando uma instância chega a esta atividade, o motor verifica se a instância também está presente em alguma

outra atividade, se estiver, a instância deve aguardar até que todas as subinstâncias alcancem o mesmo ponto. Quando isto ocorrer, a instância, já reagrupada, seguirá para a próxima atividade. Este tipo de atividade pode ter diversas transições de entrada, normalmente mais de uma, mas pode ter apenas uma atividade de saída.

2.3.7 Standalone

Atividades do tipo **standalone** são representadas por hexágonos. Este tipo de atividade não faz parte de fluxo algum e podem ser executadas a qualquer momento. São usadas, geralmente, para manter dados do processo: adicionar, remover ou modificar ítens, entre outras funções que não envolvam fluxo. Muitos processos podem ser definidos como um conjuntos de atividades deste tipo, se não existir um relacionamento entre as atividades do processo. Porém o mais comum, é que processos contenham um fluxo principal e um conjunto auxiliar de atividades standalone. As atividades isoladas não podem ter transições de entrada e saída.

2.4 Propriedades das Atividades

Existem duas propriedades das atividades que afetam a forma como o fluxo se desenvolve, são elas: (1) interatividade; (2) autoroteamento.

2.4.1 Interatividade

No Galaxia, as atividades podem ser automáticas ou interativas. As atividades interativas são as atividades que solicitam alguma informação para o usuário, normalmente através de um formulário. Após submeter a informação, a atividade é completada. As atividades automáticas são executadas pelo motor Galaxia, sem qualquer interação externa e frequentemente estão ocultas ao usuário.

2.4.2 Autoroteamento

Normalmente quando uma atividade é completada, deseja-se que o motor do workflow envie a instância, automaticamente, para a próxima atividade. Isto chama-se autoroteamento. Porém,

existem ocasiões em que deseja-se que a instância não mude de estado. Neste caso, o usuário poderá editar as informações muitas vezes antes de decidir que a instância realmente vá para a próxima atividade.

Algumas regras para a representação gráfica de fluxos de processos:

- Setas de transição saindo de uma atividade de roteamento automático são pretas;
- Setas de transição saindo de uma atividade de roteamento não automático são vermelhas;
- Atividades interativas têm a borda azul;
- Atividades não interativas têm a borda verde.

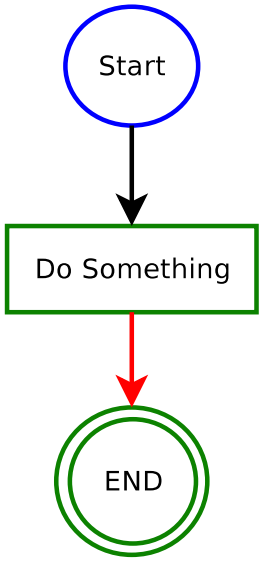


Figura 2.2: Exemplo de representação de atividades de acordo com a interatividade e do auto-roteamento.

2.5 Processo Exemplo

A Figura 2.3 mostra o gráfico de um processo, o *Music CD Loan*. Este processo define um fluxo de um empréstimo de CDs. *Request Loan* é uma atividade interativa do tipo *start*. Através dela o usuário informa o CD que deseja emprestar. Após a sua execução, a instância é enviada para a atividade *Approve Loan*, interativa do tipo *switch*. Nesta atividade o usuário responsável deverá verificar se CD solicitado está disponível. Em caso afirmativo a solicitação deve ser aceita, enviando a instância para a atividade *Approved*, não interativa do tipo *normal*. Caso contrário, a solicitação deve ser rejeitada, enviando a instância para a atividade *Rejected*), também não

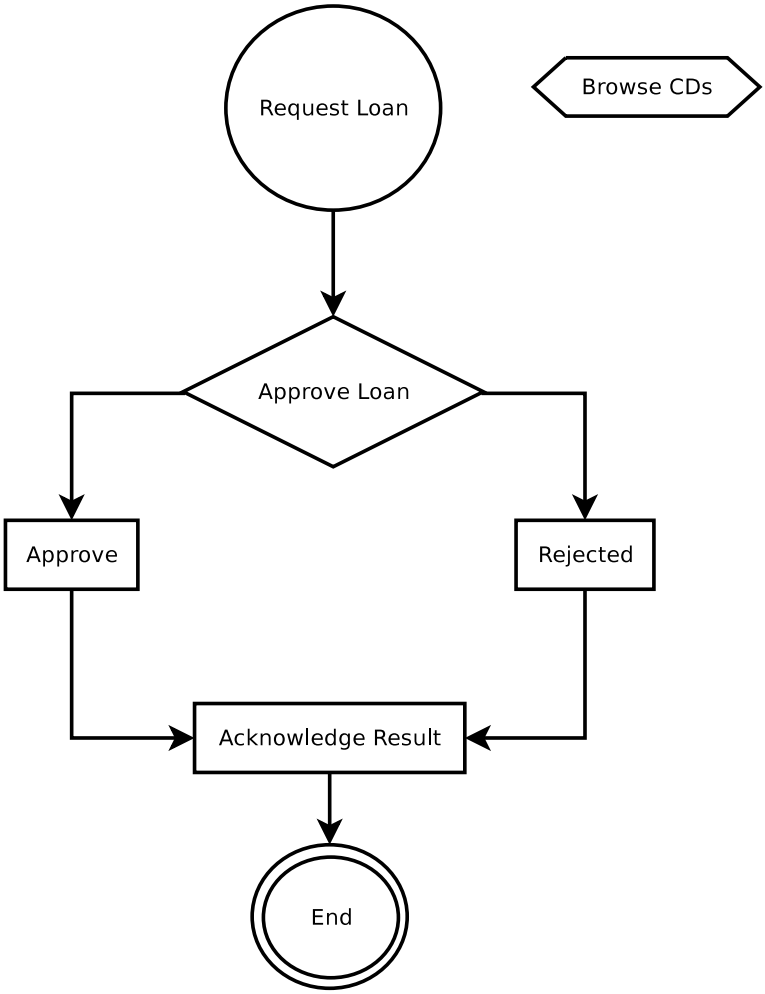


Figura 2.3: Exemplo de processo de empréstimo de CDs.

interativa do tipo *normal*. Em ambos casos a atividade é executada automaticamente e instância é enviada para a atividade *Acknowledge Result*, onde o usuário pode verificar se o seu pedido foi aprovado ou não. A atividade standalone *Browse CDs* pode ser usada pelo usuário para pesquisar o catálogo de CDs.

2.6 Módulos

O Galaxia define três módulos:

- Interface de administração;
- Interface de monitoramento;

- Interface do usuário.

2.6.1 Interface de Administração

O gerenciador de processos é um módulo usado para criar/atualizar processos. Este módulo é normalmente usado pelo administrador ou pelos designers de processo. Através desta interface, o administrador pode:

- Criar processos e versões de processos;
- Renomear e deletar atividades;
- Definir as atividades dos processos;
- Ver um gráfico das atividades do processo;
- Checar se o processo é válido;
- Ativar/desativar processos;
- Editar o código fonte das atividades (PHP) e templates (para as atividades interativas) Definir perfis e definir quais perfis tem permissão para executar quais atividades;
- Mapear perfis a usuários;
- Salvar processos (processos são salvos usando XML);
- Carregar processos a partir de arquivos XML.

2.6.2 Interface de Monitoramento

O monitor de processos é usado para controlar a execução dos processos. A lista abaixo mostras algumas funcionalidades da API do monitor de processos:

- Listar processos, atividades e o número de instâncias por atividade;
- Listar instâncias ativas e exceções;
- Percorrer a lista de instâncias e modificar suas propriedades;
- Enviar uma instância para alguma atividade;
- Assinalar ou reassinalar uma instância para um usuário; Abortar instâncias ;
- Ver estatísticas sobre os processos completados, tempo de execução, e tempo por atividade.

2.6.3 Interface do Usuário

Nesta interface o usuário pode ver os processos, os quais aquele tem permissão de executar. Esta permissão é consedida para o usuário que esteja mapeado a, pelo menos, um perfil de, pelo menos, uma atividade do processo. Esta interface também apresenta todas as instâncias que estejam paradas em alguma atividade que necessite da intervenção do usuário. Isto é chamado de “tarefa pendente”.

2.7 Resumo

Este capítulo apresenta uma introdução ao Galaxia, um motor de workflow em PHP, que pode ser usado em qualquer projeto PHP e que é fornecido juntamente com os módulos do Tiki. Galaxia pode ser usado para criar novas funcionalidades em qualquer aplicação PHP, definindo processos, onde todas as atividades relacionadas à funcionalidade estão agrupadas. Se necessário, o Galaxia pode definir o fluxo das atividades de um processo, conceito de workflow. A flexibilidade e extensão do motor abrem muitas áreas novas e interessantes para qualquer projeto PHP que use o produto.

3 INTERFACE DE USUÁRIO

O módulo de workflow possibilita que o usuário do Expresso gerencie todas as suas operações nos processos que participa. Também fornece interfaces para fácil acesso dos processos disponíveis, tarefas pendentes e acompanhamento de instâncias iniciadas pelo usuário.

Antes de começar a usar o módulo é interessante definir as preferências do usuário, conforme segue:

3.1 Preferências do Workflow

A tela onde as preferências do usuário são definidas é acessível a partir do link no menu lateral esquerdo, como pode ser visto na Figura 3.1:

Figura 3.1: Menu lateral esquerdo.

Na tela que se abre, é possível definir cinco opções:

- **Página de início**: indica qual aba será carregada por padrão ao se clicar no ícone do Workflow (cada opção é descrita na seção sobre Interfaces, logo abaixo);

- **Ordenação das tarefas pendentes**: indica qual será o critério de ordenação da aba Tarefas Pendentes (data; processo; atividade);
- **Número de itens por página**: o número de instâncias que serão exibidas na aba de Tarefas Pendentes e no de Acompanhamento;
- **Exibir página de atividade concluída**: indica se ao concluir uma atividade, deverá ser exibida uma página informativa sobre a atividade recém completada ou se o browser deve ser redirecionado para a interface do usuário (na aba de sua preferência);
- **Usar interface leve**: esta opção permite a utilização de uma interface que é carregada mais rapidamente nos navegadores (não possi imagens e poucas ações em JavaScript).

3.2 Interfaces

O Workflow disponibiliza cinco interfaces para o usuário comum do módulo, são elas: (1) Lista de Tarefas Pendentes; (2) Lista de Processos; (3) Acompanhamento de Instâncias; (4) Aplicações Externas; (5) Organograma.

3.2.1 Lista de Tarefas Pendentes

Na Lista de Tarefas Pendentes encontram-se as instâncias que podem ser acessadas pelo usuário que está logado no sistema. As instâncias podem estar disponíveis para o usuário por dois motivos:

1. A instância foi explicitamente atribuída ao usuário;
2. A instância está atribuída a um perfil do qual o usuário faz parte.

A Figura 3.2 mostra como é a interface da Lista de Tarefas Pendentes:

Nesta lista encontram-se disponíveis algumas informações acerca da instância que está com o usuário, são elas:

- **Data**: a data em que a instância chegou na atividade atual;
- **Processo**: o processo ao qual a instância pertence;
- **Identificador**: identificador da instância. Este é definido através de uma string escolhida pelo desenvolvedor do processo;

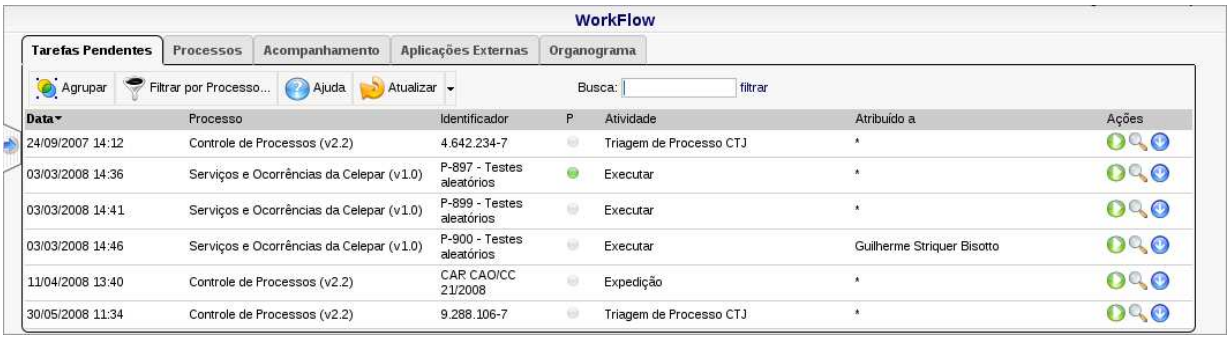


Figura 3.2: Interface da Lista de Tarefas Pendentes.

- **Prioridade (p)**: indica a prioridade da instância. Esta indicação é feita por bolinhas coloridas, onde a vermelha representa a maior prioridade e a verde a menor;
- **Atividade**: a atividade do processo na qual a instância está "parada";
- **Atribuído à**: nome da pessoa com quem está a instância. Caso esteja indicado um * (asterisco), significa que qualquer usuário do perfil relacionado à atividade pode acessar e capturar a instância;
- **Ações**: as ações disponíveis para a instância:
  - **Executar**: executa a instância. Caso esta esteja atribuída ao perfil (*), ela será capturada, ou seja, somente o usuário que a capturou poderá acessá-la;
  - **Visualizar**: visualiza dados de uma instância. Estes são: informações sobre o fluxo até o momento e propriedades daquela; item **Enviar**: remete a instância para o próximo passo do processo. Normalmente esta ação está desabilitada porque o roteamento das atividades costuma ser automático, mas podem ocorrer atividades onde é necessário que o usuário intervenha manualmente para que a instância caminhe no fluxo do processo;
  - **Liberar Acesso / Capturar**: alterna a ação entre capturar e liberar a instância. Neste último caso, o acesso é liberado para todos os usuários associados aos perfis da atividade corrente;
  - **Transformar em Exceção**: transforma a instância em exceção, não permitindo que ela seja executada. Normalmente isto é feito quando a instância não pode ser executada devido ao aguardo de algum evento;
  - **Abortar**: aborta a instância.

A princípio, para cada instância só se exibem duas ações (Executar e Visualizar) para ver as outras ações, basta clicar no botão "Mais Ações". Para ocultá-las, basta clicar novamente no mesmo botão.

O botão “Filtrar por processo” mostra somente as instância de um determinado processo (selecionado em seguida).

O botão “Agrupar” mostra a quantidade de instâncias por atividades de processo que estão na Lista de Pendências do usuário. Quando agrupadas, estará disponível um botão de nome “Desagrupar” que desagrupa as instâncias.

O botão “Ajuda” exibe o texto de ajuda da interface do usuário.

O botão “Atualizar”, se estiver ativo, irá recarregar a lista de tarefas em intervalos regulares. Se quiser desativar este recurso, clique na lista de opções do botão e escolha “Interromper atualização automática”

Na caixa de busca, coloque alguma informação que deseja procurar. O sistema irá retornar as instâncias que atendem ao argumento de pesquisa, consideradas as colunas “Identificador” e “Atividade”.

3.2.2 Lista de Processos

Na aba de Processos estão disponíveis para o usuário todos os processos cujas atividades, start ou standalone, estão associadas ao usuário através dos perfis, ou seja, o processo só aparecerá se o usuário puder iniciar um fluxo do processo ou executar alguma atividade que não esteja contida no fluxo.

A Figura 3.3 mostra como é a interface de processos:

Para verificar quais atividades estão disponíveis para o usuário, basta passar o cursor do mouse sobre o ícone do processo, as atividades disponíveis serão listadas ao seu lado. Além das atividades, também estarão disponíveis mais duas opções:

- **Gráfico do Processo**: irá abrir uma nova aba contendo um diagrama que representa o fluxo do processo;
- **Sobre o Processo**: irá abrir uma nova aba contendo informações sobre o processo e suas atividades.

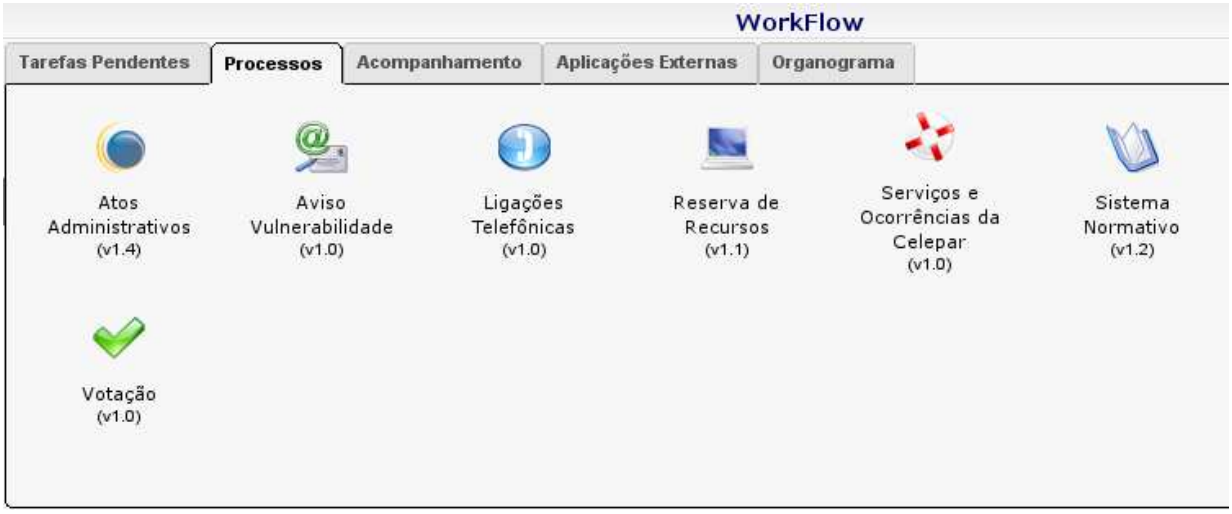


Figura 3.3: Interface da Lista de Processos.

3.2.3 Acompanhamento de Instâncias

A aba de Acompanhamento é semelhante à aba de Tarefas Pendentes, como pode ser visto na Figura 3.4.

Figura 3.4: Interface de Acompanhamento.

A diferença entre elas reside no fato de que: enquanto a Lista de Tarefas Pendentes exibe somente as instâncias que, atualmente, podem ser acessadas pelo usuário, a aba de Acompanhamento exibe as instâncias que o usuário iniciou, mesmo que, no momento, elas estejam sendo executadas por outros usuários.

A única ação disponível para as instâncias desta aba é de “visualizar”, que abre uma nova aba exibindo os dados de uma instância: data de início e término, prioridade, situação, proprietário e histórico do andamento da instância. Também serão listadas as propriedades, caso o usuário seja

o administrador do módulo ou do processo.

A barra de ferramentas desta aba é composta de três botões:

1. **Alternar**: quando o usuário clica neste botão, o workflow alterna entre dois estados: (1) Mostra as instâncias que o usuário iniciou e ainda não terminaram; (2) Mostra as instâncias que o usuário iniciou e já foram encerradas;
2. **Agrupar**: agrupa as instâncias por processo. Se ele clicar um dos grupos, somente as instâncias daquele serão exibidas. Quando as instâncias estão agrupadas o botão "Desagrupar" torna-se visível;
3. **Filtrar por Processo...**: Ao clicar neste botão, uma lista de processos aparecerá, e ao selecionar um deles, somente as instâncias daquele serão mostradas. A lista é composta por processos que tenham instâncias que tenham sido iniciadas pelo usuário corrente.

3.2.4 Aplicações Externas

Nesta aba, estão disponíveis aplicações que não estão dentro do Expresso Livre. Estas se autenticam utilizando o catálogo corporativo (o mesmo do Expresso Livre) podem ser acessadas sem necessidade de novo login. As aplicações serão abertas em uma nova janela.

A Figura 3.5 mostra a interface da aba de Aplicações Externas:

Figura 3.5: Interface de Aplicações Externas.

3.2.5 Organograma

Nesta aba são exibidas informações acerca do organograma da organização do usuário.

As áreas da empresa são listadas, de forma hierárquica, na lateral direita da interface, ao clicar sobre uma das áreas, os funcionários da mesma são exibidos.

A barra de ferramentas conta com seis componentes:

1. **Visualizar**: mostra opções de exibição do organograma: áreas, centro de custos, localidades, substituição de chefia, telefones úteis e vínculos empregatícios;
2. **Gráfico**: exibe imagem do organograma da organização, caso a imagem tenha sido carregada no cadastro da organização (updload);
3. **Busca**: permite que se busque por um usuário através de parte de seu nome. Nos resultados da busca, além do nome e do telefone também será exibida a sua área.
4. **Imprimir**: lista os dados dos funcionários de uma área, com layout simplificado, próprio para impressão.

Na Figura 3.6 é possível visualizar a interface da aba de Organograma.

Figura 3.6: Interface de Organograma.

4 INTERFACE DE ADMINISTRAÇÃO

Através desta interface, administradores do módulo ou de processos podem cadastrar ou manter processos existentes.

Para acessar a interface, entre no módulo de workflow, abra a aba lateral esquerda (Figura 3.1), e em seguida clique na opção **Administrar Processos**.

Uma vez dentro da interface de administração, será exibida a página de cadastramento de processos e uma lista dos processos existentes, com opções de manutenção, conforme a Figura 4.1.

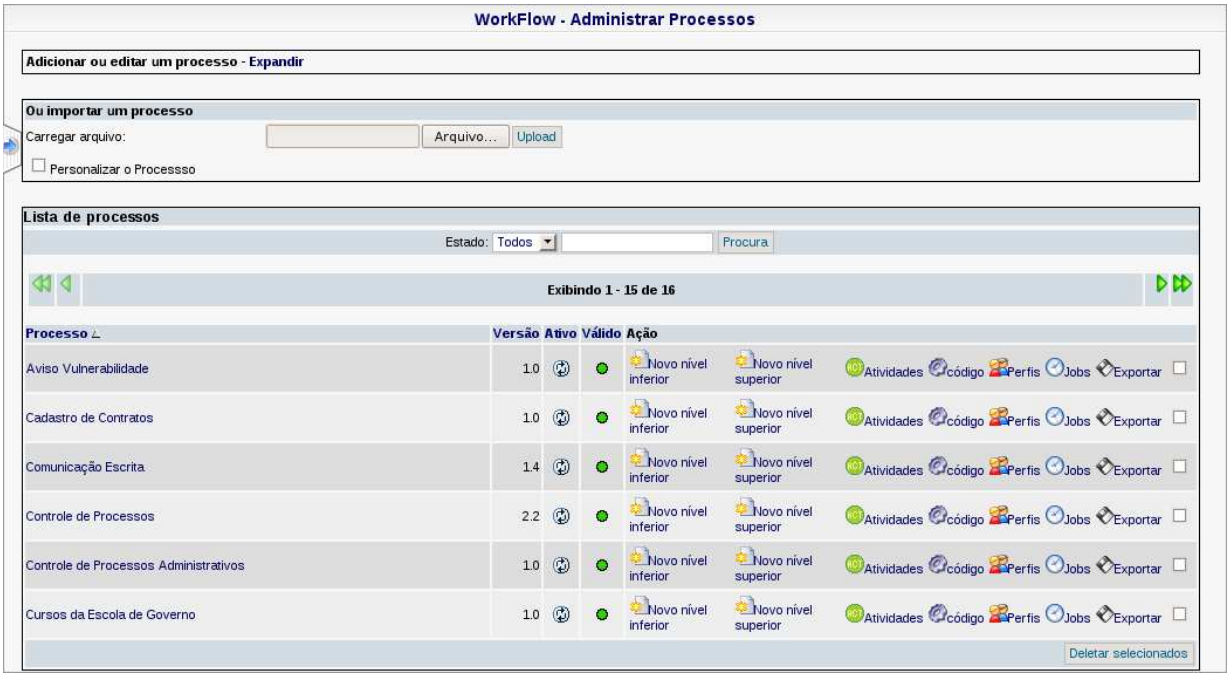


Figura 4.1: Lista de Processos para Administração.

Na parte mais ao topo da página, em *Adicionar ou Editar um Processo*, encontra-se, contraída, a interface de administração de um processo que será detalhada na seção 4.1.

Em *Ou importar um processo*, pode-se carregar o arquivo XML de um processo previamente exportado. Ao clicar em *Personalizar o processo*, mais duas opções se abrirão:

- **Nome do processo**: Nome do processo que será importado. Caso não seja colocado qualquer valor, o nome do processo será o que estiver definido no arquivo;

- **Versão do processo**: Versão do processo que será importado. Caso não seja colocado qualquer valor, a versão sera o que estiver definido no arquivo.

A parte seguinte apresenta a lista de processos. Em cada linha estão definidos os seguintes links:

- **Novo nível inferior**: Irá clonar o processo atual e incrementar o segundo dígito da versão. Por exemplo, se a versão do processo for 1.0, será criado um novo processo com versão 1.1;
- **Novo nível superior**: Idem anterior porém será incrementado o primeiro dígito do número da versão. Por exemplo, a partir de um processo 1.0 irá gerar um processo 2.0;
- **Atividades**: Possibilita registrar as atividades de um processo, suas transições e associá-las a perfis. Os detalhes serão apresentados na seção 4.2;
- **Código**: Esta interface possibilita a inclusão/edição do código das atividades, includes, imagens, javascript, css e templates;
- **Perfis**: Possibilita incluir/editar os perfis de um processo. Os perfis são elementos de estrutura usados para o controle de acesso às atividades e à eles devem ser associados usuários e/ou grupos. Pode haver mais de um perfil por processo. Por exemplo: Editor, Usuario, Gerente, etc;
- **Jobs**: Permite a criação/manutenção de jobs de um processo e também consultar os Logs gerados por sua execução. Os detalhes serão apresentados na seção 4.5;
- **Salvar**: Esta opção irá salvar a estrutura do processo em um arquivo XML. Todos os dados da estrutura serão salvos, exceto o mapeamento de usuários/grupos a perfis, uma vez que estes dados podem variar de uma instalação do workflow para outra. O arquivo gerado pode ser guardado como um backup, ou servir para implantar o processo rapidamente em uma outra instalação de workflow. A importação é feita através do formulário de cadastramento de processos, em uma seção específica para isso.

4.1 Processos

Ao clicar em *Adicionar ou Editar um Processo*, a interface mostrada na Figura 4.2 será apresentada. Através desta é possível criar um novo processo. Os campos do formulário de criação são definidos a seguir:

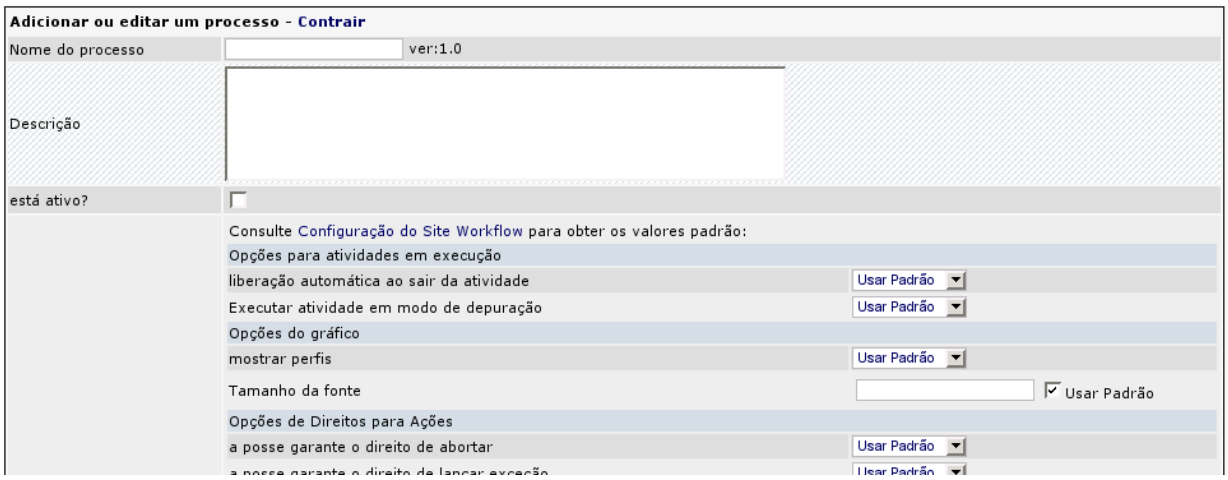


Figura 4.2: Interface de administração de Processo.

Nome do processo - o nome do processo que será criado. Por padrão assume-se que sua versão será 1.0.

Descrição - a descrição do processo. Esta informação poderá ser consultada pelo usuário do processo através do menu “Sobre o processo”.

está ativo? - indica se o processo está ativo ou não.

Valores de configuração

- **Opções para Atividades em Execução**
  - **Liberação automática ao sair da atividade**: se habilitada, esta propriedade fará com que a atividade, ao ser encerrada, saia da posse do usuário e vá para o perfil;
  - **Executar atividade em modo de depuração**: define se erros PHP serão exibidos na tela (navegador). O padrão é ocultá-los. Durante o desenvolvimento é interessante ligar esta propriedade, mas quando o processo passar para homologação, ou produção, deve ser desligada.
- **Opções Gráficas**
  - **Mostrar Perfis**: define se, no gráfico do processo, os perfis de uma atividade serão exibidos (entre colchetes);
  - **Tamanho da Fonte**: o tamanho da fonte utilizada neste gráfico.
- **Opções de Direitos para Ações** - estas opções, se habilitadas, estarão visíveis na aba de tarefas pendentes, na coluna de ações, quando o usuário clicar no botão “mais ações”.

  - **a posse garante o direito de abortar**: define se a posse de uma instância deste processo garante o direito de abortá-la;
  - **a posse garante o direito de lançar exceção**: define se a posse de uma instância deste processo garante o direito colocá-la em exceção;
  - **a posse garante o direito de liberar**: define se a posse de uma instância deste processo garante o direito de liberá-la, ou seja, deixar para que qualquer usuário do perfil possa capturá-la;
  - **perfil garante o direito de abortar**: define se um usuário no perfil associado à atividade de uma instância garante o direito de abortá-la;
  - **perfil garante o direito de liberar**: define se um usuário no perfil associado à atividade de uma instância garante o direito de liberá-la;
  - **perfil garante o direito de lançar exceção**: define se um usuário no perfil associado à atividade de uma instância garante o direito de colocá-la em exceção;
  - **desabilitar ações avançadas**: desabilita as opções de "liberar acesso", "transformar em exceção", "abortar" e "monitorar" acessíveis na aba de Tarefas Pendentes.

- **Opções de banco de dados**: Na instalação do módulo, são criadas tabelas com prefixo `egw_wf_` no esquema `public` do banco `Expresso`. Estas tabelas são exclusivas do módulo workflow e são utilizadas durante o seu funcionamento: como controle das instâncias, atividades, perfis, etc. Além do banco `Expresso` também é criado, pelo instalador, o banco `Workflow`, para armazenamento dos dados dos processos. Caso o desenvolvedor queira fazer uso deste banco, deverá criar um schema para o seu processo, bem como o seu administrador. As informações para acesso à esse banco devem ser informadas nos campos *Nome do banco de dados*, *Usuário* e *Senha* desta seção;
- **Opções de Visualização**
  - **Altura do frame de Visualização**: define, em pixels, o altura do iframe para visualização de instância na aba de tarefas pendentes (padrão $180px$). Aumente este valor caso algum processo possua muita informação para ser exibida.
- **Opções de Log**
  - **Nível de Log**: define o nível de log que será exibido quando, dentro do código do processo, for utilizada a classe *wf_log*. Os níveis vão de *Emergência*, para erros críticos até o nível *Debug* para depuração de código.
- **Agente de Correio SMTP**
  - **Profile do correio smtp**: o profile de agente smtp que será utilizado para enviar e-mails. Recomenda-se a utilização do profile já configurado para envio de e-mails do módulo de e-mail;

– **Assinatura do correio smtp**: texto a ser anexado ao final das mensagens enviadas. (Não utilizado);
– **Prefixo de link local do correio smtp**: o prefixo que será utilizado pela engine na construção de links locais (onde não é especificado `http://`);
– **Depurar correio smtp**: se habilitada, mostra informações sobre o envio do email a cada passo executado pelo agente de correio.

4.2 Atividades

Nesta interface, o desenvolvedor administra o fluxo do seu processo, registrando as atividades, transições e perfis de acesso para cada atividade.

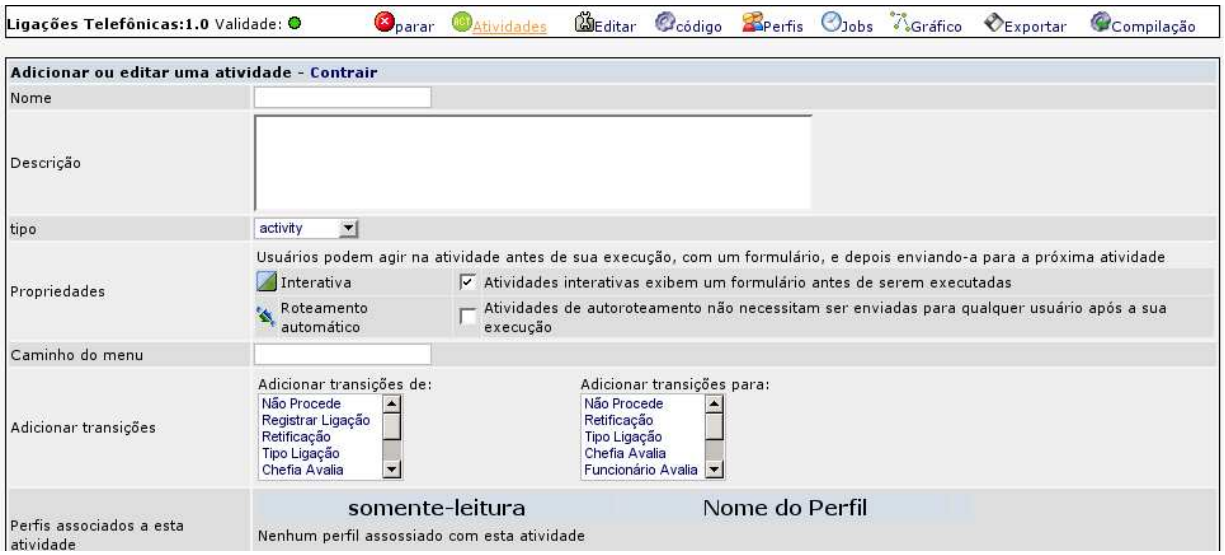


Figura 4.3: Interface de administração de Atividades.

Nome - nome da atividade. Este aparecerá no menu do processo para as atividades do tipo start e standalone e para qualquer quando aparecer na aba de acompanhamento, por este motivo, é de fundamental importância que ele seja explicativo. Em cada processo o nome deve ser único e pode conter quaisquer caracteres, como espaços ou caracteres acentuados. Para definir o nome de uma atividade usa-se verbos no infinitivo, pois elas indicam ações. p.ex: executar, emitir, compor, revisar.

Descrição - descrição do que a atividade faz. Esta informação poderá ser consultada pelo usuário do processo através do menu “Sobre o processo”.

Tipo - o tipo da atividade. Pode ser: start, end, activity, switch, split, join, standalone e view. O significado de cada um deles pode ser encontrada no seção 2.3.

Propriedades

- **interativa**: quando marcado indica que a atividade requer alguma interação com o usuário, caso contrário ela será executada automaticamente;
- **Roteamento automático**: quando marcado indica que a atividade deve passar para a próxima atividade automaticamente ao ser completada, caso contrário deve ser passada para a próxima atividade somente quando o usuário solicitar.

Caminho do Menu - esta propriedade é utilizada principalmente para a geração de submenus para uma atividade. Os níveis de submenu são delimitados pelo caractere "/". Por exemplo, supondo que temos uma atividade de nome "Cadastrar Clientes". Se a propriedade "Caminho do menu" estiver vazia, a atividade será listada no nível zero do menu. Se "Caminho do menu" possuir o valor "Cadastro", a atividade estará listada da seguinte maneira: "Cadastro... → Cadastrar Clientes". Se "Caminho do menu" possuir o valor "Cadastro/Cliente", a atividade estará listada da seguinte maneira: "Cadastro... → Cliente... → Cadastrar Clientes". Como pode ser observado, a propriedade "Caminho do menu" não deve terminar com "/" e, nos caminhos até chegar à atividade, são incluídas reticências (...).

Um outro uso que esta propriedade possui é o de ocultar uma atividade do menu. Neste caso, basta colocar o caractere "!" no valor da propriedade. Esta funcionalidade é particularmente útil quando se faz necessário desenvolver um processo sem fluxo. Já que a engine não permite a ativação de processos com este formato, cria-se as atividades start e end, sem código, e uma transição entre elas, ocultando-se a primeira.

Adicionar transições - adiciona transições de ou para a atividade que está sendo cadastrada.

Perfis associados a esta atividade - listagem dos perfis que têm permissão para executar a atividade.

Adicionar perfil - interface para associação do perfil à atividade. Pode-se adicionar um perfil previamente cadastrado, na coluna *Usar perfis existentes*, ou cadastrar um novo perfil, na coluna *Adicionar novo perfil*. Pode-se associar mais de um perfil à uma mesma atividade.

O cadastro de novos perfis podem ser feitos na interface *Adicionar Perfis de Processos* feita para este fim. Veja os detalhes na seção 4.3.

Agentes assinalados para esta atividade - nesta área estão listados os agentes associadas à atividade.

Adicionar agente - os agentes são como addons para as atividades do Workflow. O mais utilizado é o "mail_smtp" cuja função é enviar e-mails de acordo com eventos da atividade (início da atividade, fim da atividade, etc.).

Usuário Padrão - é o usuário que receberá a atividade caso não seja especificado outro explicitamente.

4.3 Perfis

A interface de administração de perfis, mostrada na Figura 4.4, é usada para cadastrar e/ou editar os perfis de um processo. Esta também é usada para mapear os perfis aos usuários ou grupos.

A primeira parte, *Adicionar ou editar um perfil*, apresenta dois campos:

- **Nome**: o nome do perfil (que deve ser único no processo);
- **Descrição**: a descrição do perfil.

Para se adicionar um perfil, basta preencher estes dois campos e clicar em "Novo". Para editar um perfil, deve-se, primeiro, clicar no nome do perfil na lista "Perfis do processo", então as informações serão carregadas nos campos próprios para serem modificadas. Neste caso deve-se clicar no botão "Salvar" para que as modificações tenham efeito permanente.

A segunda parte, *Perfis do processo*, lista os perfis do processo em edição. Através dela é possível removê-los ou editá-los.

A terceira parte, *Mapear usuários/grupos a perfis*, pode-se associar usuários ou grupos aos perfis. Para isto, selecione o perfil ao qual os usuários/grupos serão adicionados. No lado "Usuários/Grupos"adicione os ítens desejados e então clique no botão "mapear".

Na última parte, estão listados os mapeamentos do processo. Para removê-los, basta selecioná-los e então clicar em "Deletar selecionados".

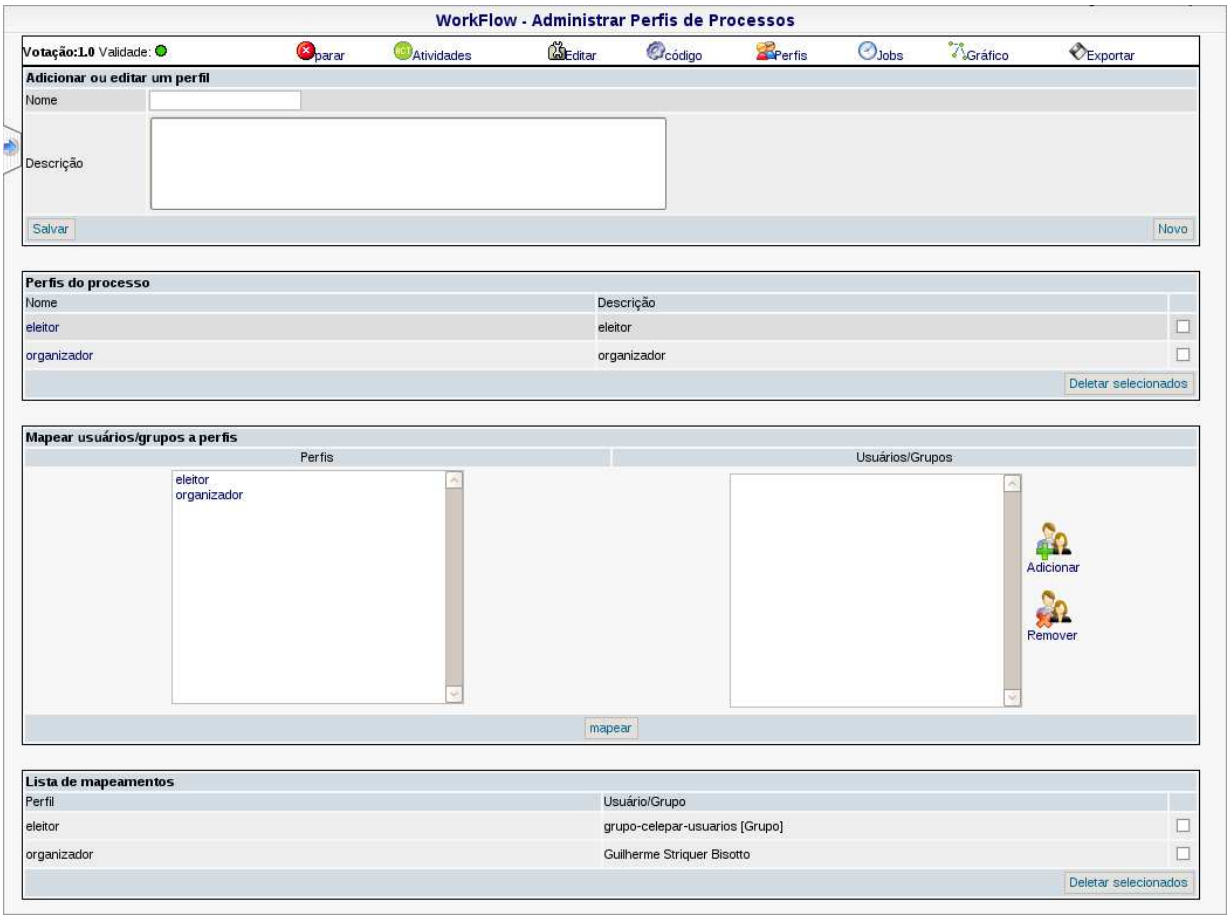


Figura 4.4: Interface de administração de Perfis.

4.4 Edição de Código

Esta interface foi desenvolvida para facilitar a edições do código do processo. Porém não há a necessidade de utilizá-la, já que todos os arquivos estão disponíveis no sistema de arquivos do servidor. Para editá-los basta acessar o diretório onde estão localizados e abrí-los utilizando o editor de texto/código de sua preferência. p.ex: Eclipse, Vim.

Ela apresenta quatro abas: **Atividades** - contém os arquivos PHP criados automaticamente ao se adicionar novas atividades (interativas ou não) ao processo. Estes são os arquivos executados diretamente pela classe `run_activity` e estão localizados no diretório `/home/expressolivre/workflow/NOME_NORMALIZADO_XX/code/activities`.

Includes - contém arquivos PHP das classes que são instanciadas na execução das ativi-

dades. Estes contém as classes de controle e modelo e estão localizados no diretório `/home/expressolivre/workflow/NOME_NORMALIZADO_XX/code`. O arquivo `shared.php` é criado automaticamente, junto com o processo. Ele contém a inclusão de todos os fontes do processo.

Templates - nesta aba encontram-se os arquivos de template do processo. Quando se cria uma atividade interativa, além do arquivo PHP que é disponibilizado na aba "Atividades", é também criado um arquivo TPL, um template Smarty semelhante a um arquivo HTML, para esta atividade. Estes arquivos estão localizados em: `/home/expressolivre/workflow/NOME_NORMALIZADO_XX/code/templates`.

Resources - nesta aba estão os arquivos que não são interpretados no servidor mas são visíveis pelo navegador do usuário, tais como figuras, códigos JavaScript e arquivos CSS. Estes arquivos estão localizados em:
`/home/expressolivre/workflow/NOME_NORMALIZADO_XX/resources`.

O nome normalizado processo, mostrado nas localizações dos arquivos de código, é definido da seguinte forma: a partir do nome do processo, os espaços são substituídos por "_", o resultado é concatenado com "_" seguido da versão do processo sem ponto. Por fim são removidos os caracteres acentuados. Exemplo:

```
Nome: Ligações Telefônicas
Versão: 1.1
Nome Normalizado: Ligaes_Telefnicas_11
```

4.5 Jobs

Nesta interface é possível administrar os Jobs e os Logs que eles geram. Na Figura 4.5 podemos ver a listagem de Jobs e ações relacionadas a eles.

Através desta interface é possível executar as seguintes ações sobre Jobs:

- **Criar**: basta clicar no botão "Novo" da interface. Ao salvar um Job, automaticamente será criado um arquivo modelo do Job;
- **Editar**: para editar um Job, basta clicar no ícone que contém uma folha de papel e um lápis. Se o nome do Job for trocado, o arquivo PHP do Job será renomeado e, a Engine

Figura 4.5: Interface de administração de Jobs.

tentará trocar o nome da classe. Caso a troca não seja possível, será enviada um alerta para o desenvolvedor informando que a troca precisa ser feita manualmente;

- **Remover**: Para remover um Job, basta clicar no figura de "proibido" (ícone vermelho) e, confirmar a exclusão. Ao remover um Job, o arquivo que o implementa também é excluído do sistema de arquivos. Além disso, os Logs produzidos pelo Job também são apagados;
- **Ativar/Desativar**: Para ativar um Job, basta clicar no "X" vermelho do lado esquerdo do seu nome. Quando ele estiver ativado, haverá um "V" verde no lugar do "X", e para desativá-lo basta clicar sobre este símbolo.

Para criar ou editar um Job faz-se uso da interface apresentada na Figura **??**.

Figura 4.6: Interface de edição de Jobs.

A propriedades desta são descritas a seguir:

Nome - o nome do Job. Este deve ser único em cada processo. Esta propriedade também define o nome da classe e do arquivo de código que serão gerados para a sua execução. Para a geração do nome da classe, o nome informado é normalizado. Esta normalização é ligeiramente diferente daquela usada no nome dos processos e atividades. Exemplo de conversão:

```
Nome: Avaliar Solicitação
Classe: AvaliarSolicitacao
Arquivo: class.job.AvaliarSolicitacao.inc.php
```

Descrição - a descrição do que o Job faz.

Ativo - indica se o Job está ativo ou não. Quando inativo não será executado mesmo que esteja agendado.

Data de Início - indica a data a partir da qual o Job será válido para execução.

Hora da Execução - a hora em que o Job será executado.

Tipo de Data - os Jobs são bem versáteis quanto à sua repetição. Esta propriedade define o critério repetição. Existem três as possibilidades:

- **Data Absoluta**: ideal para Jobs que possuem um intervalo de execução bem definido. e.g.: executar o Job a partir do dia 07/05/2008 e repetir a cada 3 dias;
- **Dias da Semana**: para Jobs que têm sua execução baseada em dias da semana. e.g.: a partir do dia 14/04/2008, executar o Job às segundas, quartas e sextas com repetição a cada duas semanas;
- **Data Relativa do Mês**: para Jobs que devem ser executados em relação ao número de dias restantes para o fim do mês. e.g.: executar o Job a cada 6 meses, no último dia do mês.

O intervalo mínimo para execução de Jobs é de 1 minuto (disponível na repetição “Data Absoluta”). Não possui limite de intervalo máximo.

Dependendo do “Tipo de Data” selecionada, outros campos podem surgir:

- **Dias da Semana**: são apresentados os dias da semana para que sejam selecionados aqueles em que o Job deve ser executado;

- **Data Relativa do Mês**: é necessário informar quando o Job deve ser executado informando a quantidade de dias para o fim do mês.

Repetição do Job - indica se o Job possui repetição ou não.

Campos de Repetição - são campos onde se informa a periodicidade em que o Job é executado.

4.5.1 Logs

O Log é o resultado de cada execução de um Job. Sua listagem pode ser acessada através do ícone de um documento contendo ítens.

O Log pode indicar quatro resultados: desconhecido, erro, falha e sucesso. Veja a Figura 4.7.

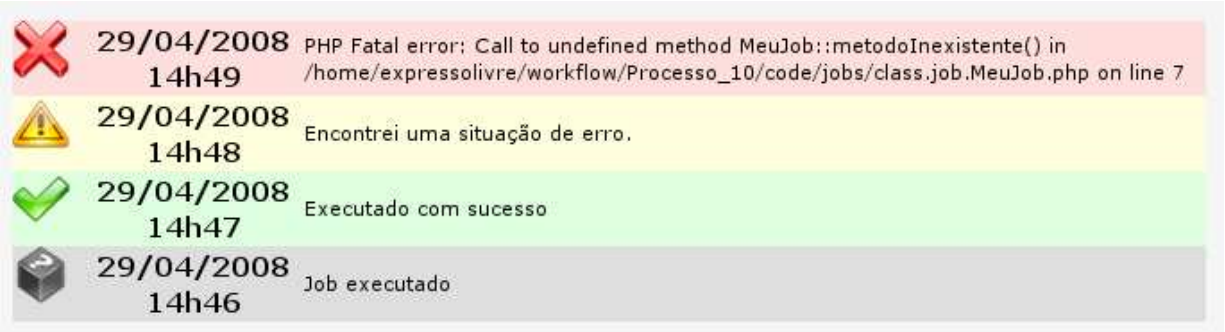


Figura 4.7: Logs de um Job.

A descrição dos status dos Logs da Figura 4.7 (de cima para baixo) é vista a seguir:

- Erro - ocorre quando o Job possui algum erro em seu código. O erro é salvo no Log;
- Falha - ocorre em dois casos: (1) o executor de Job encontra algum problema que impede sua execução; (2) o desenvolvedor chamou o método indicando que houve falha;
- Sucesso - ocorre quando o usuário chama o método que indica sucesso;
- Desconhecido - quando não ocorrem erros ou falhas detectáveis pelo executar ou gerenciador de Jobs e, o desenvolvedor não fez nenhuma chamada que indique sucesso ou falha.

4.5.2 Execução de Jobs para Teste

Na interface de administração de Jobs, é possível executar um Job para testar seu funcionamento. Para isto, basta clicar no figura da engrenagem (ícone azul).

Após a execução de um Job (acionada a partir da interface), são exibidas até três informações:

- Mensagens: informações sobre a execução do processo (e.g. tempo de execução, etc.);
- Saída Default: strings enviadas para a saída padrão. Normalmente enviadas por `echo`, `print`, `print_r`, etc.;
- Saída de Erro: erros capturados durante a execução do Job.

Um exemplo de resultado da execução de um Job (em modo de teste), pode ser visto na Figura 4.8

Figura 4.8: Resultado do teste de execução de um Job.

O código do Job que gerou a execução da Figura 4.8 pode ser visto abaixo:

```
class MeuSegundoJob extends JobBase
{
  public function run()
  {
    /* enviado antes do erro. Vai para a saída padrão */
    echo "minha saída\nantes do erro";

    /* código que vai causar erro */
    funcaoInexistente();
  }
}
```

5 INTERFACE DE MONITORAMENTO

Através da interface de monitoramento, um usuário com atribuição de monitor, pode administrar as instâncias dos processos sob sua responsabilidade e realizar diversas operações sobre elas.

Para acessar a interface, entre no módulo de workflow e abra a aba lateral esquerda (Figura 3.1) e em seguida clique na opção “Monitores” na seção “Workflow Monitoramento”.

Uma vez na interface de monitoramento, será exibida uma página com uma lista de processos que podem ser monitorados (Figura 5.1).

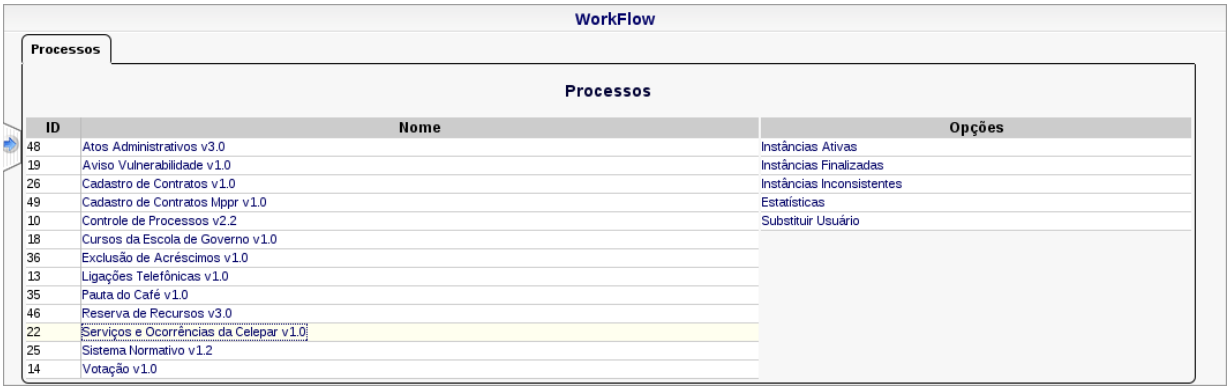


Figura 5.1: Listagem de processos na Interface de Monitoramento.

Ao clicar sobre o nome do processo desejado, uma lista de opções será mostrada. Nesta lista estarão disponíveis os seguintes links:

- Instâncias Ativas;
- Instâncias Finalizadas;
- Instâncias Inconsistentes;
- Estatísticas;
- Substituir Usuário.

5.1 Instâncias Ativas

Lista todas as instâncias do processo, exceto as finalisadas, e para cada uma delas, informa: o seu Id, a atividade atual, o identificador, a prioridade, o usuário responsável, o status e as

propriedades da instância. A lista de instâncias pode ser vista na Figura 5.2.

Instâncias Ativas

Adicionar Filtro | Filtrar | Enviar E-mail **Total de Instâncias:** 2846

1 2 3 4 5 6 próximo

ID	Atividade	Identificador	Pri.	Usuário	Status	Ações
99212	Executar	P-31962 - Ambientes - Servidor	0	Perfil: SOS_EXECUTOR_53	ativa	propriedades visualizar
99207	Distribuir	P-31960 - Ambientes - Estação	0	Perfil: SOS_DISTRIBUIDOR_89	ativa	propriedades visualizar
99206	Executar	P-31959 - Aplicações	0	Garten Nack	ativa	propriedades visualizar
99203	Executar	P-31958 - Apoio técnico a eventos	0	Alzemar Venancio	ativa	propriedades visualizar
99202	Executar	P-31957 - Apoio técnico a eventos	0	Alzemar Venancio	ativa	propriedades visualizar
99201	Distribuir	P-31956 - Ambientes - Estação	0	Perfil: SOS_DISTRIBUIDOR_89	ativa	propriedades visualizar
99200	Executar	P-31955 - Telecomunicações e infraestrutura lógica e elétrica para redes	0	Lilian Aparecida de O Piotto	ativa	propriedades visualizar
99198	Executar	P-31954 - Apoio técnico a eventos	0	Elson Baladeli	ativa	propriedades visualizar
99197	Executar	P-31953 - Serviço de Entrega	0	Maria Aparecida Soares	ativa	propriedades visualizar
99195	Distribuir	P-31952 - Administração de Usuários	0	Perfil: SOS_DISTRIBUIDOR_64	ativa	propriedades visualizar
99194	Executar	P-31951 - Serviço de Entrega	0	Maria Aparecida Soares	ativa	propriedades visualizar
99193	Executar	P-31950 - Ambientes - Servidor	0	Perfil: SOS_EXECUTOR_53	ativa	propriedades visualizar
99192	Executar	P-31949 - Serviço de Entrega	0	Maria Aparecida Soares	ativa	propriedades visualizar

Figura 5.2: Listagem de instâncias ativas na Interface de Monitoramento.

Id - número identificador da instância. Este id é utilizado pela engine para diferenciar unicamente cada uma daquelas.

Atividade - a coluna atividade, mostra em qual atividade se encontra cada instância do processo. O administrador pode mudar a atividade atual simplesmente clicando em sobre ela. Ao clicar, é apresentada uma combobox com todas as atividades do processo, então basta o administrador escolher qualquer uma e depois clicar no botão “ok” para confirmar a mudança. Deve-se utilizar este recurso com cuidado, para não comprometer a integridade da instância, enviando-a para um ponto do fluxo, onde não teria condições de prosseguir. O mais comum é enviar a instância para a atividade anterior a atual.

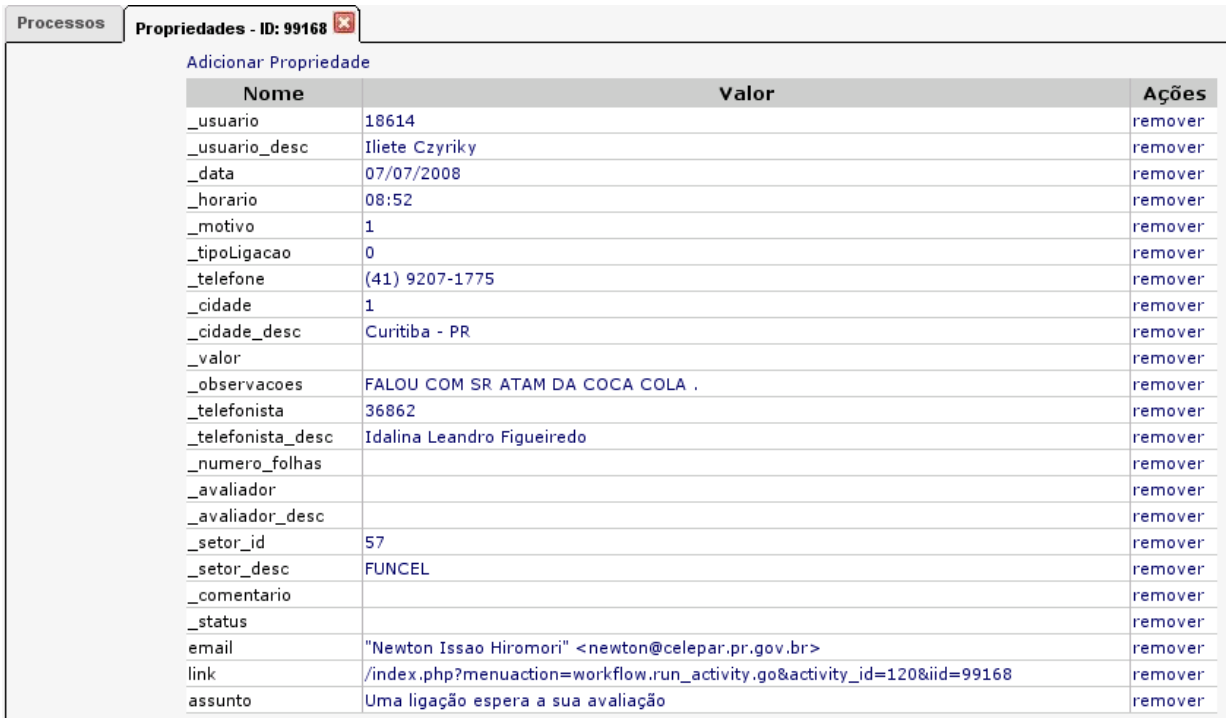
Identificador - nesta coluna, o administrador pode incluir ou alterar o identificador da instância. O identificador é um atributo da instância que pode ser utilizado pelo desenvolvedor para identificá-la de forma mais amigável. Este dado também será apresentado na caixa de entrada para auxiliar o usuário final na identifição daquelas. Por exemplo: num processo de ligações telefônicas, pode ser o número do telefone discado, ou em um processo de trâmite de documentos, o número do processo.

Prioridade - clicando-se em cima desta coluna pode-se alterar a prioridade de uma atividade de uma instância. A mudança é imediata e visivel na caixa de entrada do responsável. Os valores variam entre 0 e 4, onde este último identifica as mais urgentes.

Usuário - a coluna usuário mostra o usuário responsável pela atividade. Ao clicar sobre ela é mostrado um combobox com todos os usuários e grupos permitidos para aquela. O administrador pode atribuir a atividade atual para qualquer elemento listado.

Status - essa coluna mostra o estado no qual a instância se encontra: ativa, abortada, em execução ou completada. O administrador pode alterar o status da instância clicando sobre esse valor.

Propriedades - ao clicar nesta coluna abre-se uma nova aba de navegação onde pode-se ver e editar todas as propriedades atuais da instância e seus valores. O administrador também pode inserir novas propriedades ou removê-las. Veja um exemplo na Figura 5.3.



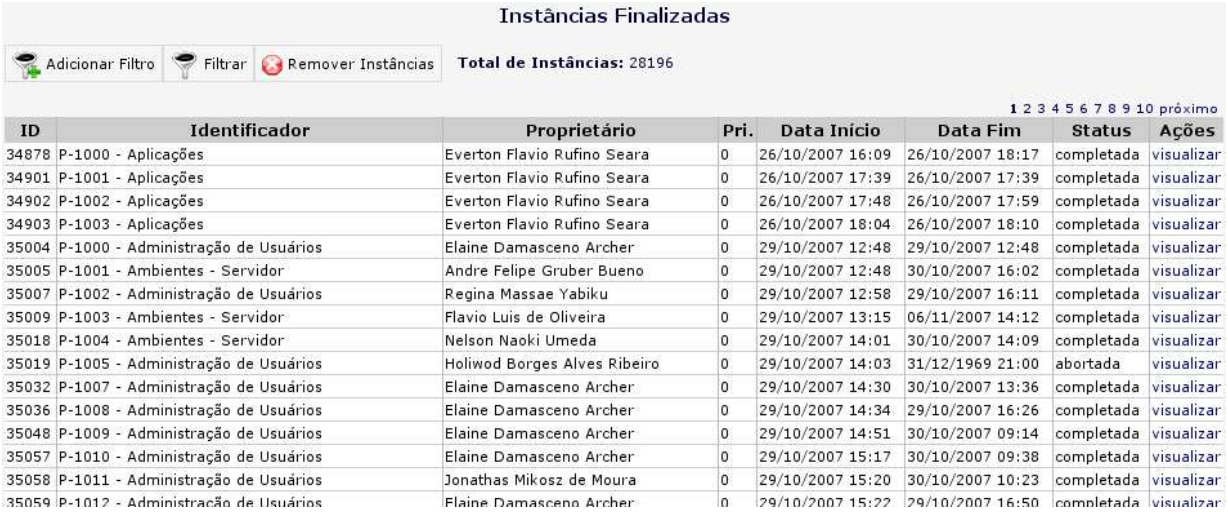
Nome	Valor	Ações
_usuario	18614	remover
_usuario_desc	Iliete Czyriky	remover
_data	07/07/2008	remover
_horario	08:52	remover
_motivo	1	remover
_tipoLigacao	0	remover
_telefone	(41) 9207-1775	remover
_cidade	1	remover
_cidade_desc	Curitiba - PR	remover
_valor		remover
_observacoes	FALOU COM SR ATAM DA COCA COLA .	remover
_telefonista	36862	remover
_telefonista_desc	Idalina Leandro Figueiredo	remover
_numero_folhas		remover
_avaliador		remover
_avaliador_desc		remover
_setor_id	57	remover
_setor_desc	FUNCEL	remover
_comentario		remover
_status		remover
email	"Newton Issao Hiromori" <newton@celepar.pr.gov.br>	remover
link	/index.php?menuaction=workflow.run_activity.go&activity_id=120&iid=99168	remover
assunto	Uma ligação espera a sua avaliação	remover

Figura 5.3: Propriedades de uma instância, vistas na Interface de Monitoramento.

Visualizar - ao clicar sobre este link irá abrir uma nova aba com os dados completos da instância. Caso exista uma atividade *view*, ela será exibida também.

5.2 Instâncias Finalizadas

Lista todas as instâncias do processo que estejam na situação “completada” ou “abortada”. Esta interface apresenta os dados somente para consulta, e não podem ser alterados. A Figura 5.4 exibe a página de monitoramento de instâncias finalizadas.



ID	Identificador	Proprietário	Pri.	Data Início	Data Fim	Status	Ações
34878	P-1000 - Aplicações	Everton Flavio Rufino Seara	0	26/10/2007 16:09	26/10/2007 18:17	completada	visualizar
34901	P-1001 - Aplicações	Everton Flavio Rufino Seara	0	26/10/2007 17:39	26/10/2007 17:39	completada	visualizar
34902	P-1002 - Aplicações	Everton Flavio Rufino Seara	0	26/10/2007 17:48	26/10/2007 17:59	completada	visualizar
34903	P-1003 - Aplicações	Everton Flavio Rufino Seara	0	26/10/2007 18:04	26/10/2007 18:10	completada	visualizar
35004	P-1000 - Administração de Usuários	Elaine Damasceno Archer	0	29/10/2007 12:48	29/10/2007 12:48	completada	visualizar
35005	P-1001 - Ambientes - Servidor	Andre Felipe Gruber Bueno	0	29/10/2007 12:48	30/10/2007 16:02	completada	visualizar
35007	P-1002 - Administração de Usuários	Regina Massae Yabiku	0	29/10/2007 12:58	29/10/2007 16:11	completada	visualizar
35009	P-1003 - Ambientes - Servidor	Flavio Luis de Oliveira	0	29/10/2007 13:15	06/11/2007 14:12	completada	visualizar
35018	P-1004 - Ambientes - Servidor	Nelson Naoki Umeda	0	29/10/2007 14:01	30/10/2007 14:09	completada	visualizar
35019	P-1005 - Administração de Usuários	Holiwod Borges Alves Ribeiro	0	29/10/2007 14:03	31/12/1969 21:00	abortada	visualizar
35032	P-1007 - Administração de Usuários	Elaine Damasceno Archer	0	29/10/2007 14:30	30/10/2007 13:36	completada	visualizar
35036	P-1008 - Administração de Usuários	Elaine Damasceno Archer	0	29/10/2007 14:34	29/10/2007 16:26	completada	visualizar
35048	P-1009 - Administração de Usuários	Elaine Damasceno Archer	0	29/10/2007 14:51	30/10/2007 09:14	completada	visualizar
35057	P-1010 - Administração de Usuários	Elaine Damasceno Archer	0	29/10/2007 15:17	30/10/2007 09:38	completada	visualizar
35058	P-1011 - Administração de Usuários	Jonathas Mikosz de Moura	0	29/10/2007 15:20	30/10/2007 10:23	completada	visualizar
35059	P-1012 - Administração de Usuários	Elaine Damasceno Archer	0	29/10/2007 15:22	29/10/2007 16:50	completada	visualizar

Figura 5.4: Listagem de instâncias finalizadas na Interface de Monitoramento.

Para cada instância listada estão disponíveis as informações:

- ID;
- Identificador;
- Proprietário (o usuário que criou a instância);
- Prioridade;
- Data de início;
- Data de término;
- Status (completada ou abortada).

Existe tambem uma coluna de ação com o seguinte link:

Visualizar - clicando sobre este link irá abrir uma nova aba com os dados completos da instância, inclusive mostrando a atividade *view* do processo, se existir.

5.3 Instâncias Inconsistentes

Lista todas as instâncias do processo que contenham algum tipo de inconsistência. Por exemplo, uma instância foi atribuída a um usuário que foi removido do catálogo, ou a instância, por algum motivo, parou em uma atividade não interativa. Veja a Figura 5.5

Instâncias Inconsistentes

Usuários Removidos

Quando uma instância está atribuída a um usuário que não está mais no LDAP.

ID	Atividade	Identificador	Pri.	Usuário	Status
228597	Avaliar atendimento	P-110617 - Aplicações	0	ID: 22974 - Leandro Haruo Nakamura (excluído)	ativa
229036	Avaliar atendimento	P-110896 - Ambientes - Servidor	0	ID: 22974 - Leandro Haruo Nakamura (excluído)	ativa

Usuários sem Autorização

Quando o usuário que está com a instância não pertence a nenhum dos perfis da atividade desta instância.

ID	Atividade	Identificador	Pri.	Usuário	Status
41981	Avaliar atendimento	P-2601 - Administração de Usuários	0	ID: 20139 - Debora Fernanda Julio de Souza (excluído)	ativa
48197	Executar	P-4664 - Ambientes - Estação	0	ID: 22298	ativa
228597	Avaliar atendimento	P-110617 - Aplicações	0	ID: 22974 - Leandro Haruo Nakamura (excluído)	ativa
229036	Avaliar atendimento	P-110896 - Ambientes - Servidor	0	ID: 22974 - Leandro Haruo Nakamura (excluído)	ativa

Instâncias Falhas

Quando uma instância encontra-se em uma atividade não-interativa.

ID	Atividade	Identificador	Pri.	Usuário	Status
116775	end	P-41265 - Aplicações	0	*	ativa
117012	Splitter	P-41365 - Administração de Usuários	0	*	ativa

Instâncias Finalizadas Vinculadas a uma Atividade

Quando uma instância finalizada ou abortada ainda está vinculada a uma atividade.

Nenhuma ocorrência encontrada.

Figura 5.5: Listagem de instâncias inconsistentes na interface de monitoramento.

Atualmente são rastreados 4 tipos de inconsistências:

Usuários Removidos - a instância foi atribuída a algum usuário antes deste ser removido do catálogo. Neste caso não houve tempo para o usuário enviar a instância para outra atividade ou liberar o acesso para o perfil.

Usuários Sem Autorização - engloba a primeira inconsistência, porém neste caso, algum usuário pode ter recebido uma instância e logo depois ele foi removido dos perfis associados à atividade corrente.

Instâncias Falhas - por algum problema a instância parou em uma atividade não interativa. Neste caso deve-se corrigir o código falho do processo e acessar o monitoramento para forçar o

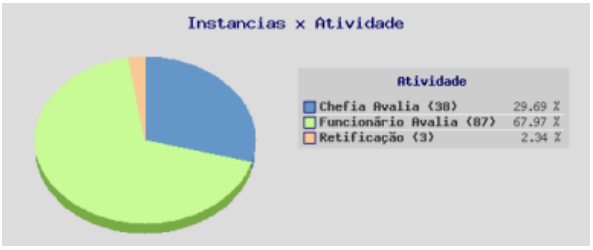
envio da instância para a próxima atividade.

Instâncias Finalizadas Vinculadas a uma Atividade - neste, uma instância foi abortada ou finalizada, mas por algum motivo ela ainda está associada a uma atividade do meio do fluxo.

5.4 Estatísticas

Mostra graficamente dados estatísticos sobre as instâncias. As Figuras 5.6 (a)–(d) mostram os tipos de gráficos estatísticos produzidos pelo módulo.

(a) Quantidade de instâncias geradas a cada mês.



(b) Instâncias de acordo com a atividade em que se encontram.

(c) Instâncias de acordo com seu usuário atual.

(d) Instâncias de acordo com o status em que se encontram.

Figura 5.6: Gráficos estatísticos produzidos na Interface de Monitoramento.

5.5 Substituição de Usuário

Nesta opção de monitormento é possível transferir as instâncias que estão de posse de um usuário para outro. Isto é útil para o caso em que um funcionário está afastamento as suas instâncias do processo devam passar para a posse de outro usuário, que dará continuidade a elas.

A operação de substituição é efetuada em três passos.

Primeiro Passo - selecionar o usuário que está com a posse e o usuário que irá receber as instâncias. Este passo está exemplificado na Figura 5.7:

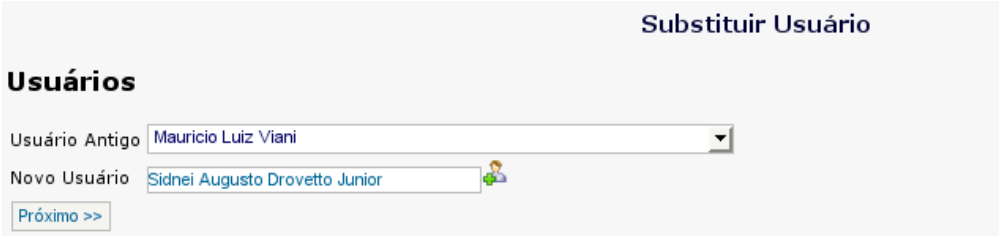


Figura 5.7: Substituição de Usuário - passo 1: selecionar o usuário que será substituído e o substituto.

Segundo Passo - escolher qual a atividade do processo será utilizada para selecionar as instâncias que sofrerão a troca de usuário. Pode-se escolher todas.

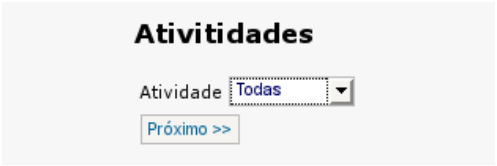


Figura 5.8: Substituição de Usuário - passo 2: filtrar por atividades.

Terceiro Passo - aqui será feita uma verificação de consistência para saber se o usuário que receberá as instâncias pertence aos perfis exigidos. Caso, ele não pertença, será exibido um formulário que permite sua inclusão nos perfis apropriados. Quando o usuário estiver nos perfis necessários, será permitida a substituição.

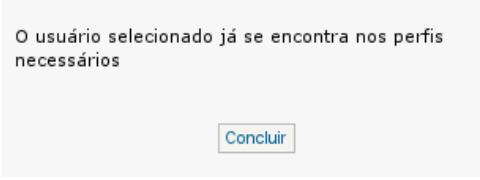


Figura 5.9: Substituição de Usuário - passo 3: associação aos perfis.

5.6 Filtro de Instâncias

Esta é uma importante ferramenta de auxílio ao monitoramento dos processos. Com ela o usuário monitor pode encontrar intâncias de acordo com critérios de seleção. A funcionalidade de filtro está disponível para as opções de Intâncias Ativas e Instâncias Finalizadas. Ao ser escolhida uma destas opções será exibida uma barra de botões para filtragem, conforme mostrado na Figura 5.10:

Figura 5.10: Barra de filtragem de instâncias.

As opções são as seguintes:

- Adicionar filtro: deve ser adicionado pelo menos um filtro de seleção. Podem ser adicionados mais filtros e o módulo irá fazer a união dos mesmos ao filtrar as instâncias;
- Filtrar: este é o botão de ação a ser acionado depois que os filtros forem adicionados;
- Total de instâncias: contador de instâncias selecionadas pelo(s) filtro(s).

A filtragem das instâncias ativas pode ser feita utilizando os seguintes critérios:

- Atividade da Instância: a atividade em que a instância está parada no momento;
- Data da Instância: data quando a instância foi criada;
- Data na Atividade: data quando a instância chegou na atividade atual;
- ID da Instância: o número sequencial único de identificação da instância;
- Identificador: o atributo que identifica a instância. Este pode variar de acordo com o processo;
- Prioridade: o grau de urgência das instâncias. Varia de 0 a 4, onde o último é o mais prioritário;
- Proprietário: o usuário que criou a instância;
- Status: a situação atual da instância, podendo ser ativa, completada, abortada, ou em espera;
- Usuário da Instância: o usuário que detém a posse da instância no momento.

Já as instâncias finalizadas podem ser filtradas de acordo com os seguintes critérios:

- Data da Instância: idem à listagem anterior;
- Data de Finalização da Instância: a data em que a instância foi finalizada;
- ID da Instância: idem à listagem anterior;
- Identificador: idem à listagem anterior;
- Prioridade: idem à listagem anterior;
- Proprietário: idem à listagem anterior;
- Status: idem à listagem anterior.

Lembrando que podem ser usados mais de um filtro ao mesmo tempo. Depois que os filtros estiverem construídos, clique no botão “Filtrar”.

5.7 Ações em Massa

É possível a execução de algumas ações em massa, ou seja, a mesma ação para todas as instâncias listadas na interface de monitoramento. Para restringir o conjunto de instâncias, pode-se utilizar os filtros. Atualmente, duas ações estão disponíveis: (1) envio de e-mail, para as instâncias ativas e; (2) remoção de instâncias, para as instâncias finalizadas.

5.7.1 Envio de E-mail

Com o envio de e-mail é possível alertar os usuários sobre situações que necessitam a sua intervenção, como por exemplo, instâncias paradas a muito tempo com determinado usuário. Primeiramente filtre as instâncias que deseja, e depois clique no envio de e-mail para compor uma mensagem de aviso. A Figura 5.11 mostra a interface para envio de e-mails:

Será exibido um texto de e-mail padrão, mas pode-se modificar o texto conforme necessário. Existem variáveis de substituição, cadeias de caracteres iniciadas e terminadas por %, que ao serem inseridas no texto serão, no momento da execução, trocadas pelos valores definidos. As variáveis disponíveis podem ser vistas na Tabela 5.1.

Ao terminar a composição do e-mail, clique no botão *Preview* para visualizar a mensagem antes de enviá-lo definitivamente.

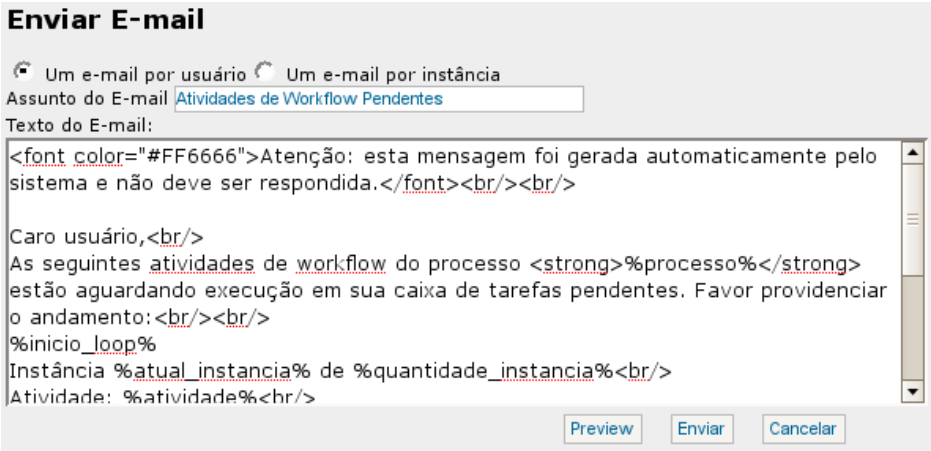


Figura 5.11: Interface para envio de e-mails.

Existe ainda uma opção adicional para indicar como as instâncias serão relacionadas no e-mail:

- Um e-mail por usuário: irá montar um e-mail único e listar dentro dele cada instância que o usuário possui. Útil caso o número de instâncias seja grande;
- Um e-mail por instância: nesta opção, para cada instância filtrada, será gerado um e-mail (um mesmo usuário pode receber vários e-mails).

5.7.2 Remover Instâncias

Esta ação está disponível somente para as instâncias finalizadas. Tem como objetivo a remoção das instâncias finalizadas que satisfazem o critério de filtragem, se este for empregado.

Um dos motivos que que podem exigir a exclusão de instâncias é a liberação de espaço no banco de dados, já que, além das instâncias, seus ítens de trabalho, que podem ocupar grande espaço em disco, também são excluídos.

Variável	Significado
%atividade%	O nome da atividade atual
%usuario%	O nome do usuário que está com a instância
%processo%	O nome do processo
%identificador%	O identificador da instância
%tempo_atividade%	A duração (até o momento) da atividade atual
%tempo_instancia%	A duração (até o momento) da instância
%inicio_atividade%	O início da atividade atual
%inicio_instancia%	O início da instância
%quantidade_instancia%	A quantidade de instâncias com o usuário
%atual_instancia%	A instância atual (número em relação à quantidade)
%link%	Link para a execução da instância (abre em nova janela)
%url%	Somente a URL para a execução da instância
%prioridade%	A prioridade da instância

Tabela 5.1: Lista de variáveis que podem ser utilizadas no envio de e-mails.

6 METODOLOGIA

Nesta seção são apresentadas as diretivas recomendadas para o desenvolvimento de processos no Workflow do Expresso Livre.

6.1 Documentação Mínima de Projeto

Obviamente, não se pode desenvolver um sistema sem antes passar pela etapa de projeto. No caso do workflow, esta premissa também é válida. Mesmo para processos simples é necessário seguir uma metodologia de desenvolvimento de sistemas, sob pena de cair em erros de implementação por falta de planejamento. Portanto, antes de começar a codificar o seu processo, atente para as recomendações descritas a seguir.

Recomenda-se a utilização de alguma metodologia de projeto e também alguma ferramenta de conteúdo, para facilitar o registro da documentação, tickets de pendências, e programação das etapas do projeto. Por exemplo, no ambiente Workflow da Celepar/PR é utilizada a linguagem de modelagem UML e o software Trac.[5]

[5] *Trac* disponível em `http://trac.edgewall.org`

6.2 Padrões de Codificação PHP

Os padrões de codificação abaixo são muito importantes para a legibilidade do código. Procure seguir fielmente o que está descrito neste documento, para que o seu código fique padronizado e possa ser lido por outras pessoas que venham a dar manutenção em seu processo.

Tags PHP:

- Usar <?php ?> para delimitar código PHP, pois é a forma mais portável de incluir código PHP em diferentes sistemas operacionais e configurações.

Identação e Comprimento de Linha:

- Usar tabulação equivalente a 4 espaços;
- É recomendado que as linhas tenham, no máximo, entre 75–85 caracteres.

Comentários:

- Comentários (blocos) de documentação PHPDoc (ver seção 6.3) são obrigatórios;
- Comentários adicionais explicativos, no meio do código, também são recomendados;
- Comentários estilo multi-linha (`/* */`) e linha única (`//`) são recomendados, ao passo que estilos Perl/shell (`#`) devem ser evitados;
- No comentário de elementos como classes ou funções, evitar repetir a classe do elemento na descrição. Por exemplo, no comentário de uma função, evitar incluir a palavra "função".

Incluindo Código:

- Includes somente podem ser inseridos no arquivo `shared.php`. Caso seja absolutamente necessário incluir algum arquivo fora deste local, atentar para estas recomendações:
  - Usar `require_once` ao incluir incondicionalmente arquivos de classe;
  - Usar `include_once` ao incluir condicionalmente arquivos de classe;
  - `require_once` e `include_once` não são funções, portanto parênteses não devem envolver o nome do arquivo.

Convenções de Nomenclatura:

- Classes:
  - Devem ter nomes descritivos. Evitar usar abreviações. Nomes de classes devem começar com uma letra maiúscula;
  - Exemplos: `Classe`, `MinhaClasse`.

- Métodos, Funções e Atributos:
  - Devem usar o padrão *camel-case* (Java). p.ex.: `metodo`, `meuMetodo`, `meuAtributo`;
  - Métodos da camada de controle e que fazem parte do MVC devem ser verbos no infitivo. p.ex.: `imprimir`, `enviar`, etc.
  - Métodos da camada de modelo e que fazem parte do MVC devem ser verbos sucedidos pela palavra `Action`. p.ex.: `imprimirAction`, `enviarAction`, etc.
  - Membros privados de classes são precedidos de um underscore. p.ex: `_adicionar`, `_adicionarAction`, `_atributo`, etc.
- Constantes:
  - Devem ser definidas, de preferência, no arquivo `shared.php`;
  - Escritas em letras maiúsculas, com underscores separando palavras;
  - Podem ser prefixadas com o nome da classe/pacote onde elas são usadas;
  - As constantes true, false e null são exceções e devem ser escritas com letras minúsculas.
- Variáveis Globais:
  - Se o projeto precisar definir variáveis globais, seus nomes devem iniciar com um underscore seguido pelo nome do projeto e outro underscore. Por exemplo, o pacote PEAR usa uma variável global chamada `$_PEAR_destructor_object_list`.

Nome de Arquivos:

- De atividades: mesmo nome das atividades, adicionando a extensão `.php`. Exemplo: `Compor.php`, `Escrever.php`, `Finalizar.php`;
- De templates: mesmo nome das atividades e respectivas ações, adicionando a extensão `.tpl`. p. ex: `Compor.tpl`, `Imprimir.tpl`, `Visualizar.tpl`;
- De classes: segue o formato `class.nomeclasse.nomesuperclasse.inc.php\verb`. p.ex: `class.compor.controller.inc.php` ou `class.imprimir.model.inc.php`.

Estruturas de Controle:

- Incluem as instruções `if`, `for`, `while`, `switch`, etc. Segue abaixo um exemplo:

```
<?php
if ((condition1) || (condition2))
{
  action1;
}
elseif ((condition3) && (condition4))
{
```

```
  action2;
} else{
  defaultaction;
}
?>
```

- Instruções de controle devem ter um espaço entre a palavra-chave e a abertura de parênteses, para distingui-los de chamadas de funções;
- É recomendado sempre usar chaves mesmo em situações onde elas são tecnicamente opcionais, pois aumentam a clareza da leitura e diminui a chance de erros lógicos serem introduzidos quando novas linhas forem adicionadas;
- Para instruções `switch`:

```
<?php
switch (condition) {
  case 1:
    action1;
    break;

  case 2:
    action2;
    break;

  default:
    defaultaction;
    break;
}
?>
```

Chamadas de Função::

- Devem ser feitas sem espaços entre o nome da função e os parênteses, e o primeiro parâmetro; espaços entre vírgulas e cada parâmetro, e sem espaço entre o último parâmetro e o parênteses e o ponto-e-vírgula. Segue um exemplo:

```
<?php
$var = foo($bar, $baz, $quux);
?>
```

- Como mostrado acima, deve haver um espaço entre a variável e o operador de atribuição e entre o operador e a chamada a função. Pode ser introduzidos mais espaços para aumentar a legibilidade;

```
<?php
$short         = foo($bar);
$long_variable = foo($baz);
?>
```

Definição de Funções:

- Argumentos com valores padrão aparecem ao final da lista de argumentos. Exemplo:

```
<?php
function fooFunction($arg1, $arg2 = '')
{
  if (condition) {
    statement;
  }
  return $val;
}
?>
```

Definição de Classes:

- A chave de abertura fica na linha inferior ao do nome da classe e a de fechamento fica na linha seguinte da última linha de código:

```
<?php
class Classe
{
  /* código PHP */
}
?>
```

- Definir apenas uma classe por arquivo `.php`;
- Classes com métodos contendo argumentos string esperando valores pré-definidos devem declarar atributos internos escritos em maiúscula para funcionarem como constantes com os valores válidos, visando minimizar erros de digitação do argumento e aproveitar o *autocomplete* dos editores para membros de classe. Por exemplo:

```
<?php
class Classe
{
  var $VERSION = "1.0";
```

```
  function showVersion($var)
  {
    echo $var;
  }

  function Classe()
  {
    $this->showVersion($this->VERSION);
  }
}
?>
```

- Deve-se seguir esta sequência de definições em uma classe:
  - Declaração de constantes;
  - Declaração de atributos;
  - Declaração de construtores;
  - Declaração de métodos.

6.3 Documentação de Código

O phpDocumentor é o mais utilizado para auto-documentação da linguagem PHP. Similar ao Javadoc, e desenvolvido em PHP, pode ser usado da linha de comando ou através de uma interface web para criar documentação profissional para códigos-fonte PHP. phpDocumentor oferece suporte para relacionar documentações, incorporando documentos de usuário como tutoriais e a criação de código-fonte com destaque visual com referência cruzada para documentação genérica do PHP.

phpDocumentor usa um sistema completo de templates para mudar os comentários do seu código-fonte em formatos mais legíveis e, portanto, úteis. Este sistema permite a criação de documentações de fácil leitura em 15 estilos prontos em versões HTML, PDF, CHM e XML. Você também pode criar seus próprios templates para obter um visual mais próximo do seu projeto.

O phpDocumentor é capaz de documentar automaticamente comandos `include`, `define`, funções, páginas procedurais, classes, seus atributos e métodos. Todos podem conter as tags padrão.

Dentre as tags disponíveis no phpDocumentor, as mais relevantes no contexto dos processos de workflow são:

- `@author [nome_autor]`: autor do elemento atual;
- `@copyright [string]`: informações sobre direitos autorais;
- `@deprecated [version/info string]`: elementos abandonados que não devem mais ser usados, podendo ser removidos em futuras versões;
- `@example [arquivo]`: incluir um arquivo externo de exemplo com coloração de sintaxe;
- `@filesource`: similar a `@example`, porém voltado a arquivos internos;
- `@ignore`: evita a documentação de um elemento;
- `@internal [descrição]`: define descrição de elemento como privada, interna ao projeto e que não deve ser exibida na documentação final;
- `@link [URL link text, URL, URL, URL...]`: mostra um link dentro da documentação;
- `@see [file.ext|elementname|class::methodname()|class::$variablename| functionname()|function functionname Número ilimitado de valores separados por vírgulas]`: link para a documentação de um elemento;
- `@since [version/info string]`: versão a partir da qual um elemento passou a integrar um pacote;
- `@todo [information string]`: modificações a serem feitas;
- `@tutorial [package/ subpackage/ tutorialname.ext #section.subsection description]`: mostra um link para a documentação de um pacote;
- `@uses [file.ext|elementname|class::methodname()|class::$variablename| functionname()|function functionname]`: descrição de como o elemento é usado;
- `@version [versionstring]`: versão de um elemento.

Exemplo:

```
/**
 * Gerencia o processo básico
 *
 * @author Carlos Eduardo Nogueira Gonçalves
 * @package Workflow
 * @license http://www.gnu.org/copyleft/gpl.html GPL
 */
```

Diferentemente das tags normais, as tags *inline* podem ser usadas em qualquer lugar dentro do bloco de comentários. São elas:

- `{@link}`: exibe um link para uma URL ou a documentação de algum elemento:
  `{@link URL description}`
  `{@link element description}`
- `{@tutorial}`: exibe um link para um tutorial:
  `{@tutorial package/ subpackage/ tutorialname.ext #section.subsection description}`
- `{@source}`: exibe o código-fonte de uma função ou método na descrição longa:
  `{@source startline number of lines}`
- `{@inheritdoc}`: usada para que as subclasses herdem a descrição detalhada de suas superclasses:
  `{@inheritdoc}`

Exemplo 1:

```
/**
 * Text with a normal @tutorial tag
 * @tutorial phpDocumentor/phpDocumentor.pkg
 */
```

Exemplo 2:

```
/**
 * inline tags demonstration
 *
 * this function works heavily with {@link foo()} to rule the world. If
 * I want to use the characters "{@link" in a docblock, I just
 * use "{@}link." If I want the characters "{@*}" I use "{@}*}"
 */
```

Elementos procedurais: são considerados elementos procedurais comandos `include`, `define`, funções, e páginas procedurais.

Bloco de comentário em nível de página: um bloco de página pode ter qualquer uma das tags padrão, e deve obrigatoriamente:

- Ser o primeiro bloco em um arquivo;
- Conter uma tag `@package`.

Comandos `include/require/include_once/require_once`: o phpDocumentor extrai o nome do arquivo e tenta ligar a documentação para o respectivo arquivo se possível. Comandos include podem apenas ter as tags padrão. O phpDocumentor tentará localizar o arquivo incluído na lista de arquivos analisados, encontrando-o fará uma ligação para a documentação do arquivo.

Comandos `define`: o bloco de comentários de uma declaração `define` poderá conter qualquer tag padrão do phpDocumentor, acrescido da tag:

- `@name [nome_var]`: especifica um apelido para uma página procedural ou variável global.

Exemplo:

```
/**
 * Prefixo das tabelas do workflow
 * @name GALAXIA_TABLE_PREFIX
 */

define('GALAXIA_TABLE_PREFIX', 'egw_wf_');
```

Variáveis globais:

- `@global (obrigatória): [datatype $globalvariablename, datatype description];`
- `@name [nome_var]`: especifica um apelido para uma página procedural ou variável global.

Exemplo:

```
/**
 * Special global variable declaration DocBlock
 * @global integer $GLOBALS['_myvar']
 * @name $_myvar
 */

$GLOBALS['_myvar'] = "myval";
```

Declaração de funções:

- `@global [datatype $globalvariablename, datatype description];`
- `@param [datatype $paramname description]`: Parâmetro de função ou método;
- `@return [datatype description]`: Tipo de retorno de função ou método;

- `@staticvar [datatype description]`: Uso de uma variável estática dentro de uma função ou método;
- `inline {@source}`.

Exemplo:

```
/**
 * Includes files from the process folder.
 * @param string $file_name File's name to be included.
 * @return void
 * @license http://www.gnu.org/copyleft/gpl.html GPL
 * @access public
 */

function wf_include( $file_name )
```

Classes:

- `@package [nome_pacote]`: agrupa classes, funções e constantes
- `@subpackage [nome_subpacote]`: agrupa classes, funções e constantes de um pacote
- `@category [nome_categoria]`: organizar o pacote do elemento documentado
- `@static`: classe ou método estático
- `@abstract`: documenta uma classe abstrata, atributo ou método
- `@license [URL name of license]`: mostra um link para a URL da licença

Os blocos de comentários são herdados pelas classes filhas, variáveis e métodos. As regras para herança são:

- As tags `@author`, `@version` e `@copyright` são automaticamente herdadas a não ser explicitamente especificado em contrário;
- Embora as tags `@package` e `@subpackage` sejam herdadas automaticamente, é recomendado que sejam usadas explicitamente para todas as classes para evitar problemas de conflitos;
- Em caso de ausência de descrição breve, ela será herdada;
- Em caso de ausência de descrição longa, ela será herdada;
- Em caso de haver uma descrição longa e ainda se quiser herdar a descrição da superclasse, use `inline {@inheritdoc}`.

Exemplo:

```
/**
 * Criptografia simples e segura baseada em funções hash
 * @author Marc Wöhlken, woehlken@quadracom.de
 * @author Carlos Eduardo Nogueira Gonçalves
 * @version 1.0
 * @license http://www.gnu.org/copyleft/gpl.html GPL
 * @package Workflow
 */
```

Exemplo usando descrição breve e descrição longa:

```
/**
 * short desc
 *
 * long desc
 * @package test
 * @author me
 * @version 1.0
 * @abstract
 * @copyright never
 */
```

Atributos: o bloco de comentário de atributos pode conter qualquer tag padrão e obrigatóriamente a tag `@var`:

- `@var [datatype $varname descrição]`: um atributo;
- `@access [private, protected, public]`: controle de acesso para um elemento

Exemplo:

```
/**
 * @var string $hash_key Versão embaralhada da chave de criptografia
     fornecida pelo usuário
 * @access public
 */

var $hash_key;
```

Métodos:

- `@access [private, protected, public]`: controle de acesso para um elemento;

- `@global [datatype $globalvariablename, datatype description]`;
- `@param [datatype $paramname description]`: parâmetro de função ou método;
- `@return [datatype description]`: tipo de retorno de função ou método. Caso retorne `mixed` especificar quais os tipos possíveis na descrição;
- `@static`: classe ou método estático;
- `@staticvar [datatype description]`: uso de uma variável estática dentro de uma função ou método;
- `@final`: método que não deve ser sobrescrito nas subclasses;
- `inline {@source}`.

Exemplo:

```
/**
 * Usado para definir a chave de criptografia e descriptografia
 * @param string  $key  Chave secreta usada para criptografia e
    descriptografia
 * @param boolean $base64 Usar codificação base64
 * @return void
 * @access public
 */

function setKey($key, $base64 = true)
```

6.4 Padrões de Nomenclatura para Bancos de Dados

Normas Gerais:

- Use letras maiúsculas para palavras reservadas (sintaxe) SQL.
- Use letras minúsculas para elementos de negócio (particulares do projeto em desenvolvimento):
  - eliminar a dúvida sobre qual a "caixa" correta assim como outros erros relacionados;
  - aumenta a velocidade de escrita e exatidão;
  - diferencia nomes de tabelas e campos da sintaxe com caixa alta do SQL.
- Separe palavras e prefixos com "_" (*underscore*), nunca use espaços:
  - melhora legibilidade (ex: `nome_livro` e `nomelivro`);

  - evita a necessidade de envolver nomes com colchetes (ex: `[nome livro]` ou `'nome livro'`);
  - maior independência de plataforma.

- Evite usar números;
- Procure identar os comandos SQL, principalmente se os mesmos forem extensos;
  - Melhora a legibilidade do código.

Exemplo sem identação:

```
SELECT filial_id, produto_id, pro_descricao, pro_valor_unitario
FROM produto WHERE filial_id = 2 AND pro_valor_unitario > 100;
```

Exemplo com identação:

```
SELECT filial_id, produto_id, pro_descricao, pro_valor_unitario
FROM   produto
WHERE  filial_id          = 2
AND    pro_valor_unitario > 100;
```

Tabelas

- Escolha nomes sem ambiguidade, curtos, não usando mais que duas palavras:
  - distingue tabelas facilmente;
  - facilita nomear campos únicos assim como tabelas de metadados.
- Use nomes no singular, nunca plural:
  - promove consistência com a nomenclatura de campos de chave primárias e tabelas de metadados;
  - garante ordenação alfabética de uma tabela antes de suas tabelas de metadados ou relacionadas;
  - evita confusão de regras de plural do português ou inglês;
  - estrutura SQL mais "gramatical" (e.g. `SELECT activity.activity_name` ao invés de `SELECT activities.activity_name`).
- Evite nomes com acrônimos, abreviados ou concatenados:
  - provê arquitetura auto-documentável;
  - facilita tanto para desenvolvedores quanto para não-desenvolvedores lerem e entenderem.
- Prefixe as tabelas de metadados (*lookup tables*) com o nome das tabelas a que elas se relacionam:

  - agrupa tabelas relacionadas (ex: `activity_status`, `activity_type`, etc);
  - evita conflitos de nomes de tabelas de metadados de diferentes entidades.

- Para uma tabela associativa ($n : m$), concatene o nome das duas tabelas envolvidas:
  - expressa o propósito de composição da tabela;
  - deve ser evitado quando houver muitas tabelas com junção para as mesmas entidades originais.

- Crie comentários para a tabela e para as colunas:
  - facilita a compreensão do propósito da tabela
  - explica o conteúdo presente em uma coluna.

Campos/Colunas

- A chave primária deve ter o nome da tabela com o sufixo `_id`:
  - permite que a chave primária seja deduzida ou lembrada a partir apenas do nome da tabela (e.g., chave primária da tabela `produto` seria `produto_id`;
  - consistência com o nome da chave primária;
  - evita a necessidade de usar apelidos (*alias*) na programação;
  - para tabelas que possuem mais de um campo compondo a chave primária, essa regra não se aplica, sendo que os campos poderão continuar tendo o sufixo `_id`, porém o seu nome que antecede tal sufixo terá que ser outro diferente do nome da tabela. e.g., tabela `imobilizado`, `PK = patrimonio_id + ano_id`;
  - quando a chave primária é composta por campos `FK`, os mesmos permanecerão com os nomes dos campos `PK` das tabelas relacionadas. e.g., tabela `item_nota_fiscal`, a `PK` seria `nota_fiscal_id` e `item_nota_fiscal_id`, onde o campo `nota_fiscal_id` é a `FK` que vem de outra tabela e o campo `item_nota_fiscal_id` é da própria tabela em questão, e ambos juntos formam a `PK`.

- Chaves estrangeiras devem ter o mesmo nome das chaves primárias às quais elas se referem:
  - faz com que as tabelas às quais elas se referem completamente óbvio;
  - se houver múltiplas chaves estrangeiras se referenciando a uma mesma tabela, prefixe o campo da chave estrangeira com um adjetivo descritivo apropriado (`adjetivo_nome_campo`, e.g., `titular_funcionario_id`).

Restrições/Constraints

- O nome da chave primária deverá ser formado pelo nome da tabela, acrescido do sufixo `_pkey`, e.g., tabela `reserva_sala`, chave `reserva_sala_pkey`;

- O nome das chaves estrangeiras deverão ser formados pelo nome da tabela + o sufixo `_fknn`, onde `nn` é uma sequência começando em `01`, e.g., `reserva_sala_fk01`;
  - O objetivo em gerar as chaves estrangeiras em sequência é simplificar a escrita;

Índices

- O nome de um índice deverá ser formado pelo nome da tabela + campo indexado + sufixo `_idx`, e.g., `ocorrencia_servico_id_idx`.

7 ARQUITETURA

O desenvolvimento de processos de Workflow utiliza a arquitetura *Model - View - Controller* (MVC), já consagrada entre desenvolvedores, ela propõe a organização do código em camadas. Cada uma irá limitar-se exclusivamente àquilo que se propõe a fazer, mantendo tudo em seu contexto a fim de tornar transparente a localização de cada parte do código.

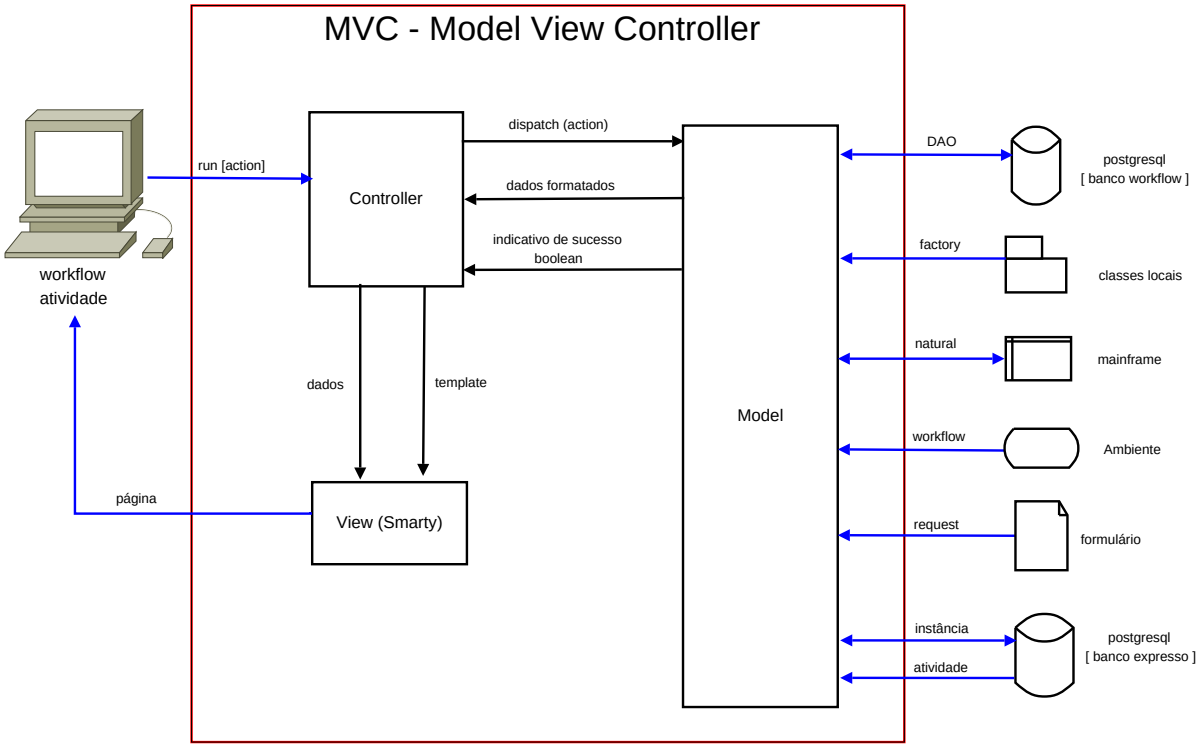


Figura 7.1: Interface de mapemaneto de Perfis.

A Figura 7.1 esquematiza a arquitetura MVC empregada em uma atividade qualquer de um processo. Vale dizer que todas as atividades do processo deverão seguir este modelo de implementação como forma de padronizar o código e facilitar a manutenção.

Observa-se que as três camadas do MVC estão representadas no esquema. O ponto inicial de ação é a execução da atividade por parte do usuário, representado pelo ícone do computador e o método `run()` que ativa a camada de controle.

A camada de controle irá processar a requisição `run()` e tentará identificar qual a ação que está sendo solicitada pelo usuário. Feita esta identificação, o método adequado da própria camada *Controller* será acionado através do `dispatch()`. Quando for a primeira execução da atividade,

nenhuma ação será requisitada, então a camada *Controller* irá acionar o método `__default()` da própria *Controller*. Uma vez no método adequado, métodos da *Model* serão chamados para executar a ação solicitada.

Uma vez que o processamento passe para a camada *Model*, a mesma terá acesso a recursos disponibilizados pelo módulo Workflow, que são:

- **DAO**: é um objeto de conexão com o banco de dados (que significa *Data Access Object*). Este objeto já estará configurado para utilizar o banco de dados do processo. Uma vez solicitada a conexão, o objeto estará pronto para receber comandos SQL, como por exemplo *queries* de consulta, ou comandos de inserção de dados;
- **Factory**: este é um objeto especial responsável por criar outros objetos sob demanda do usuário. Por exemplo, com a *factory* podemos requisitar a criação de um objeto de organograma, e com ele ter acesso aos dados do organograma da organização;
- **Natural**: objeto de conexão com o ambiente *Mainframe* para execução de programas Natural;
- **Workflow**: este objeto contém informações sobre o ambiente de execução da atividade corrente, como por exemplo, qual o usuário está logado;
- **Request**: contém os dados submetidos pela atividade. Por exemplo, se a ação a ser tratada pela *Model* for um salvamento com inserção no banco de dados, neste objeto estarão os dados preenchidos pelo usuário no formulário da atividade;
- **Instance**: objeto da instância atual;
- **Activity**: objeto da atividade atual.

O retorno da camada *Model* será um indicativo de sucesso ou falha da operação. A camada *Controller* irá tratar este retorno para tomar uma decisão sobre o que fazer. Geralmente após o retorno da camada *Model*, alguma informação será exibida para o usuário, com uma mensagem de sucesso, ou uma página de resultado de pesquisa, etc. Neste ponto, a camada *Controller* irá carregar na camada *View* os dados a serem exibidos e acioná-la passando qual o *template* a ser utilizado para exibição dos dados.

Assim fecha-se o ciclo e a atividade fica no aguardo de uma ação do usuário para novamente acionar a camada *Controller*.

7.1 Código das Atividades

Para cada processo de workflow, o módulo irá reservar um espaço em disco para gravar os arquivos que formarão o código do processo. O local onde estes dados estão armazenados está representado pelo esquema abaixo. O desenvolvedor poderá optar por editar os arquivos de código diretamente no sistema de arquivos (no disco do computador, seguindo o modelo abaixo) ou então utilizar a interface web de codificação.

```
+ /home/expressolivre/workflow
  + nome_normalizado_do_processo
    + code
      + activities
      + jobs
      + templates
    + graph
    + logs
    + resources
    + smarty
      + cache
      + compiled
```

O primeiro passo para codificar uma atividade é definir o seu código básico, que por conveniência será sempre o mesmo. Os arquivos de atividade ficam no diretório `activities` e para cada uma irá existir um arquivo PHP. Por exemplo, se tiver uma atividade no fluxo com o nome de "Cadastrar Fornecedor", teremos um arquivo com o nome de `Cadastrar_Fornecedor.php`. Ao criar o arquivo da atividade o módulo irá executar uma normalização no nome, substituindo espaços pelo caracter *underscore* e irá suprimir caracteres especiais.

Exemplo de código de uma atividade:

```
<?php
/**
 * Atividade Cadastrar do Processo
 * @author Fulano
 * @version 1.x
 * @package Nome_do_Processo
 */
```

```
/* instanciação da classe de camada de controle */
$application = new CadastrarController(new CadastrarModel($env), $env);
$application->run($_REQUEST['action']);
?>
```

Na primeira linha do código acima são iniciadas as camadas *Model* e *Controller* da atividade. Na segunda linha, o método `run()` da camada de controle é acionado e é passado como parâmetro a ação a ser executada.

7.2 Camada de Controle

Por definição, a camada de controle faz o gerenciamento entre o usuário (camada *View*) e a camada *Model* (funções do sistema). Ela deve saber apenas *quais* são as funções do sistema e não *como* implementá-las. Será responsável por receber as ações solicitadas pelo usuário, chamar a implementação da *Model* correspondente e com base na resposta, encaminhar uma interface (*View*) adequada de volta ao usuário.

No esquema MVC, empregado no módulo Workflow, a camada de controle está dividida em três níveis, cada qual implementando parte da solução, como descrito a seguir:

- Nível módulo: age sobre todos os processos de workflow;
- Nível processo: estende o nível de módulo e age sobre todas as atividades do processo;
- Nível atividade: estende o nível de processo e age somente sobre a atividade.

Começando pelo nível de módulo, ele é representado pela classe BaseController, que possui atributos e métodos de uso geral, disponíveis para todas as atividades. A classe não precisa ser codificada porque já está inclusa no pacote de código do módulo workflow, basta estendê-la.

Atributos da classe BaseController:

- `templateFile`: armazena o nome do *template* em uso pela camada *View*;
- `view`: objeto da classe Smarty que representa a camada *View*;
- `model`: objeto da classe *Model* da atividade.

Métodos da classe BaseController:

- `showForm`: troca o *template* em uso;

- `syncVars`: transfere todos os atributos da camada *Model* para a camada *View*;
- `loadViewVars`: recupera os dados produzidos pela camada *Model* e os transfere para a camada *View*;
- `assign`: define o valor de uma variável da camada *View*;
- `cancelar`: aborta a execução da atividade corrente;
- `dispatch`: executa a ação solicitada pelo usuário;
- `__default`: método abstrato a ser redefinido nas classes filhas;
- `run`: método abstrato a ser redefinido nas classes filhas.

Partindo para o nível de processo, ele é representado pela classe `Controller`, que deve ser implementada pelo desenvolvedor do processo, seguindo estas diretrizes:

- Criar um arquivo com o nome `class.controller.inc.php`;
- Armazenar o arquivo no diretório `code` da estrutura de diretórios do processo;
- Se preferir editar o arquivo pela interface web, ele fica na aba "includes" da interface de codificação;
- Definir uma classe com o nome `Controller` que estenda a classe `BaseController`;
- Definir um ou mais atributos para os *templates* da atividade;
- Implementar o método `__default()`;
- Implementar o método `run()`.

Segue um exemplo de classe *Controller* de um processo:

```
<?php
/**
 * Implementa a Classe Controller do Processo
 * @author Fulano
 * @version 1.x
 * @license http://www.gnu.org/copyleft/gpl.html GPL
 * @package Nome_do_Processo
 */
class Controller extends BaseController
{
  /**
   * @var string Template padrão da atividade
   * @access protected
   */
```

```
  var $TEMPLATEPADRAO = 'template.tpl';

  /**
   * Construtor da classe
   * @param object $model Instância da camada Model
   * @param array Configurações do ambiente MVC
   * @return object
   * @access protected
   */
  function Controller(&$model , &$env)
  {
    $this->super(&$model , &$env);
  }

  /**
   * Execucao da atividade.
   * @param $action Servico (acao) solicitado
   * @return void
   * @access protected
   */
  function run($action)
  {
    $this->dispatch($action);
  }

}
?>
```

O acionamento da camada de controle se dará pela execução do método `run()`, que por sua vez fará uso do método `dispatch()`, que executará o método de ação adequado. Por exemplo, se for requisitada uma ação "salvar", será executado o método `salvar()`, que deverá estar declarado a nível de atividade. Caso nenhuma ação seja recebida, ou o método não seja encontrado, será executado o método `__default()`.

A próxima etapa é é a implementação da classe de Controle a nível de atividade. Para isso é necessário executar os seguintes passos:

- Criar um arquivo com formato de nome seguindo o padrão `class.nome.da.atividade.controller.inc.php` no diretório `code`;
- Definir uma classe com nome no formato `AtividadeController` e que estenda a classe

`Controller` do processo;

- Implementar um método `__default();`
- Implementar um método `run();`
- Implementar um método para cada ação da atividade.

Exemplo de uma classe *Controller* de uma atividade:

```
<?php
/**
 * Implementa a Camada Controller da atividade Cadatrar
 * @author Fulano
 * @version 1.x
 * @license http://www.gnu.org/copyleft/gpl.html GPL
 * @package Nome_do_Processo
 */
class CadastrarController extends Controller
{
  /**
   * Acao padrao/inicial da atividade.
   * @return void
   * @access protected
   */
  function __default()
  {
    $this->model->defaultAction();
    $this->loadViewVars();
    $this->showForm($this->TEMPLATEPADRAO);
  }

  /**
   * Executa a ação salvar da atividade
   * @return void
   * @access protected
   */
  function salvar()
  {
    if (!$this->model->salvarAction())
    {
      $this->loadViewVars();
      $this->showForm($this->TEMPLATEPADRAO);
    }
```

```
    }
}
?>
```

Note que os métodos `run()` e `__default()` devem ser implementados, seja na classe `Controller` ou na classe `AtividadeController`, dependendo de como o desenvolvedor julgar mais conveniente. Neste caso não há a necessidade de reimplementar o método `run`, pois ele seria idêntico ao da classe pai.

É importante ressaltar que para cada ação da atividade, deverá existir um método correspondente na classe *Controller* a nível de atividade.

Analisando o método `salvar()`, vemos que o mesmo faz uso da camada *Model* para executar a operação de sistema (`salvarAction`) e caso obtenha sucesso irá carregar os dados na camada *View* e a seguir definir qual o *template* será usado para mostrá-los.

7.3 Camada de Modelo

A camada *Model* define o que o processo vai fazer e como implementá-lo, ou seja, as regras de negócio do sistema. Ela possibilita o desenvolvimento de recursos, independentemente da forma como os dados serão exibidos na interface e do momento em que serão executados. Essa camada deve saber somente como executar cada recurso do sistema.

Assim como na camada de controle, a camada *Model*, também é subdivida em três níveis:

- Nível módulo: representado pela classe `BaseModel`;
- Nível processo: representado pela classe `Model`;
- Nível atividade: representado pela classe `AtividadeModel`.

Os três níveis unem-se por herança no seguinte esquema: `BaseModel` → `Model` → `AtividadeModel`.

A classe `BaseModel` está declarada no módulo Workflow e vale para todos os processos. Nela estão codificados atributos e métodos de uso geral a serem usados pelas classes que a estenderem.

Atributos da classe BaseModel:

- `instance`: objeto da instância em execução;

- `activity`: objeto da atividade em execução;
- `DAO`: objeto da camada de abstração de banco de dados;
- `request`: array com os dados submetidos pelo formulário da camada *View*;
- `workflow`: array com as variáveis do ambiente de execução da atividade;
- `natural`: objeto para conexão com o *mainframe*;
- `viewData`: array contendo os dados a serem exibidos na camada *View*
- Contém atributos privados da camada *Model*;
- Contém atributos que representam as propriedades da instância em execução;
- Pode conter outros atributos para armazenamento de instâncias de classes de negócio (como paginação, filtros, etc);

Em especial o atributo `viewData`, seu objetivo é armazenar os dados a serem retornados para a camada de controle. Esta é a forma ideal para que um método de negócio da *Model* faça o retorno de dados.

Os atributos da instância, por padrão, deverão começar com o caracter *underscore* para diferenciá-los dos atributos privados da classe.

Métodos da classe BaseModel:

- `getAttributes`: retorna os atributos da instância (aqueles iniciados por *underscore*);
- `getRequest`: cria atributos na classe para as variáveis do array `Request`;
- `addViewVar`: adiciona uma variável ao array `viewData`;
- `getViewVar`: recupera uma variável do array `viewData`;
- `setWfProperty`: define o valor de uma variável do ambiente de execução;
- `getWfProperty`: recupera o valor de uma variável do ambiente;
- `setNameInstance`: define o valor do identificador da instância;
- `updateInstance`: transfere os atributos da classe para o objeto `instance`;
- `updateAttributes`: carrega, nos atributos da classe, os valores que estão no objeto `instance`;
- `commitInstance`: sinaliza para o módulo Workflow que a atividade atual da instância está finalizada e, que a engine pode passar para a atividade seguinte. As propriedades da instância são salvas ao se chamar este método.

Já a nível de processo, deve-se:

- Criar um arquivo com formato de nome `class.model.inc.php` e armazená-lo no diretório `code` do processo;
- Codificar a classe `Model` que estenderá a classe `BaseModel`.

Exemplo de uma classe *Model* de processo:

```
<?php
/**
 * Implementa a Classe Model do Processo
 * @author Fulano
 * @version 1.x
 * @license http://www.gnu.org/copyleft/gpl.html GPL
 * @package Nome_do_Processo
 */
class Model extends BaseModel
{
  // Aqui vai o bloco de atributos privados da classe
  /**
   * @var $variavel
   * @access protected.
   */
  var $variavel;

  // Aqui vai o bloco dos atributos de instância

  /**
   * @var $_variavel
   * @access protected
   */
  var $_variavel;

  // Aqui vai o bloco de métodos privados da classe

  /**
   * Descrição do método
   * @return array Resultado da seleção.
   * @access protected
   */
  function listar()
  {
    // comandos do método
```

```
    return $resultado;
  }

  /**
   * Valida dados do formulário.
   * @return array Resultado da validação.
   * @access protected
   */
  function verifica_erros()
  {
    // comandos de validação dos dados
    return $erro;
  }

  /**
   * Construtor da camada Model em nível de processo.
   * @param array $env Configuracao MVC.
   * @return Model.
   * @license http://www.gnu.org/copyleft/gpl.html GPL
   * @access protected
   */
  function Model(&$env)
  {
    $this->super(wf_get_env());
  }
}
?>
```

Agora, a nível de atividade valem as mesmas regras de criação de arquivo, só que o nome será no formato `class.nome.da.atividade.model.inc.php`. O arquivo deverá conter:

- Uma classe com nome no formato `AtividadeModel`, que estenda a classe `Model` do processo;
- Métodos que implementam cada uma das ações solicitadas pelo usuário e encaminhadas pela *Controller*;
- Resultados booleanos indicando sucesso ou falha na requisição feita pelo usuário.

A seguir, um exemplo de camada *Model* de uma atividade:

```
<?php
```

```
/**
 * Implementa a Camada de Modelo da Atividade Cadastrar
 * @author Fulano
 * @version 1.x
 * @license http://www.gnu.org/copyleft/gpl.html GPL
 * @package Nome_do_Processo
 */
class CadastrarModel extends Model
{
  /**
   * Implementa acao padrao da atividade Cadastrar
   * @license http://www.gnu.org/copyleft/gpl.html GPL.
   * @access public.
   * @return boolean.
   */
  function defaultAction()
  {
    // Comandos do método, por exemplo:

    $this->addViewVar("mensagem", "Digite uma mensagem.");

    return true;
  }

  /**
   * Implementa acao salvar
   * @return bool
   * @access public
   */
  function salvarAction()
  {
    if (!count($erro = $this->verifica_erros($this->request)))
    {

      $this->_mensagem = trim($this->request['mensagem']);
      $this->addViewVar("mensagem", "Sua mensagem foi salva.");

      $this->updateInstance();
      $this->setNameInstance($this->_mensagem);
      $this->commitInstance();
```

```
      return true;
    }
    else
    {
      $this->addViewVar("mensagem", "Mensagem inválida. Digite novamente.");

      return false;
    }
  }
}
?>
```

Neste exemplo estão codificados duas ações da atividade: default e salvar. Ambas serão acionadas pela camada de controle, conforme as ações forem solicitadas. O método `defaultAction()` é encarregado de preparar os dados a serem exibidos na camada de visualização quando a atividade for executada pela primeira vez. Já o método `salvarAction()` implementa o que fazer quando o usuário clicar no botão "Salvar" do formulário da atividade. Analisando mais a fundo este método, ele começa fazendo uma consistência nos dados vindos do formulário (representados pelo atributo `request`). Caso sejam válidos realiza as seguintes operações:

- Armazena os dados do formulário como atributos da camada *Model*;
- Em seguida executa o método `updateInstance()` que irá transferir para a instância os atributos da camada *Model*;
- Executa o método `commitInstance()`, que fará a atualização da instância definitivamente e encerrará a execução da atividade;
- Retorna indicativo de sucesso da operação.

Caso exista inconsistência nos dados do formulário, irá:

- Preparar os dados para serem exibidos novamente ao usuário (chamadas ao método `addViewVar`);
- Retornar indicativo de falha da operação.

Isso conclui a camada *Model* a nível de atividade.

7.4 Camada de Visualização

Corresponde à interface do sistema, onde as pessoas irão interagir com formulários para entrada de dados. Podemos também considerar como camada de visualização outras formas de interação, como as interfaces para programas (API) ou Webservices, mas não é o caso do módulo Workflow por enquanto.

Características desta camada:

- Essa camada deve saber apenas como apresentar os dados gerados pelo sistema;
- Manipulada diretamente apenas pela camada *Controller*, como um de seus atributos;
- Não deve fazer restrições de segurança ou validação de dados, que são funções da camada *Model*;
- Apresenta interface para interação (entrada de dados) com o usuário;
- Formata e exibe dados vindos da camada *Model*.

O módulo Workflow faz uso do gerenciador de *templates* Smarty, que é um conhecido projeto do mundo PHP para gerenciar a camada de visualização. Por conseqüência, não será necessário criar classes como foi feito nas camadas de controle e modelo, mas as interfaces serão contruídas como *templates* a serem executados pelo Smarty.

Os *templates* podem ser editados diretamente no diretório `templates` do modelo de estrutura de diretórios do processo, mas também podem ser editados pela interface de edição de código, na aba “Templates”.

É importante dar uma olhada no site do projeto Smarty[6] para conhecer seu potencial. Aqui nesta documentação será tratado somente o essencial.

O exemplo abaixo corresponde ao *template* `Solicitar.tpl` do processo *Music CD Loans*, que é um dos processos de exemplo que acompanham o módulo Workflow. O seu objetivo é mostrar uma lista de CDs para o usuário e colocar um botão para solicitar o empréstimo:

```
{wf_default_template}
<div id="conteudo_corpo">
  <h2>Solicitar Empréstimo</h2>
  <table class="form_tabela" width="100%" cellspacing="0" border="0" align="
      center">
```

[6] http://smarty.php.net/

```
    <tr>
      <td width="100%">
        <table id="list_tabela_clara" width="60%" cellspacing="0">
          <tr>
            <th>CD Desejado:</th>
          </tr>
          <tr>
            <td>
              <select id="cdsel">
              {section name=ix loop=$cds}
                <option value="{$cds[ix].cdid}">{$cds[ix].title}</option>
              {/section}
              </select>
            </td>
          </tr>
          <tr>
            <td>
              <input type="button" class="form_botao" onclick="dispatch('
                  Solicitar',␣document.getElementById('cdsel').options[
                  document.getElementById('cdsel').selectedIndex].value)"
                  value="Solicitar"/>
            </td>
          </tr>
        </table>
      </td>
    </tr>
  </table>
</div>
```

Os delimitadores para os comandos Smarty são as chaves (`{}`). Com isso em mente, podemos ver que o exemplo já começa com um comando Smarty, que, no caso, um plugin responsável pela inserção de cabeçalho e rodapé na página: `{wf_default_template}`.

A seguir prossegue montando uma tabela em que uma das colunas terá um campo de seleção, representado pela tag `<select>`. Para montar o conteúdo desta tag, está sendo usando o comando Smarty `{section}` que fará um *loop* sobre a variável `$cds`. O resultado será uma lista de tags `<options>`.

Depois o exemplo continua com a inserção de um botão cuja ação principal é a chamada da função JavaScript `dispatch()` que irá invocar a camada de controle passando a ação solicitada

pelo usuário. Esta função é incluída quando se adiciona o `{wf_defaul_template}`.

Concluindo, em um *template* Smarty pode-se fazer muitas coisas, e o mais importante é entender que o *template* terá comportamento dinâmico conforme forem os dados passados para ele. No exemplo anterior, vê-se uma mistura de HTML, JavaScript, CSS e comandos do Smarty. Tudo aquilo que começar com o caracter cifrão (`$`) representa uma variável que foi carregada pela camada de controle com o método `loadViewVars`. Quando o Smarty executar o *template* irá substituir as variáveis pelos seus respectivos valores, e executar os comandos. O resultado será a página a ser enviada para o usuário.

7.5 Arquivo de Configuração do Processo

Todo processo de workflow tem um arquivo `shared.php` que fica gravado no diretório `code` do processo. O objetivo deste arquivo é incluir código que seja comum a todas as atividades. Quando uma atividade for executada, o módulo Workflow incluirá o código do arquivo `shared.php` no início da atividade, de modo que seja a primeira coisa a ser executada.

Uma das utilidades do arquivo é servir de local para declarar as constantes do processo, mas a sua principal utilidade é fazer a inclusão dos arquivos de classe criados para as camadas de controle e modelo.

Exemplo de um arquivo `shared.php`:

```
<?php
/**
 * @brief Arquivo de configuração do processo.
 * @author Fulano
 * @version 1.x
 * @package Nome_do_Processo
 */

/* Início da definicao de constantes da aplicacao */
/* Fim da definicao de constantes da aplicacao */

/* Início da importacao/inclusao de classes */

/* camada de controle do processo */
wf_include('class.controller.inc.php');
```

```
wf_include('class.concluir.controller.inc.php');
wf_include('class.aprovar.controller.inc.php');
wf_include('class.solicitar.controller.inc.php');
wf_include('class.consultar.controller.inc.php');

/* camada da logica de negocios */
wf_include('class.model.inc.php');
wf_include('class.concluir.model.inc.php');
wf_include('class.aprovar.model.inc.php');
wf_include('class.solicitar.model.inc.php');;
wf_include('class.consultar.model.inc.php');

/* classes de negocio/auxiliares */

/* Fim da importacao/inclusao de classes */
?>
```

8 TUTORIAL DE DESENVOLVIMENTO DE UM PROCESSO SIMPLES

Nas seções seguintes, será abordada uma estratégia para desenvolvimento de um processo de Workflow. Utilizando esta estratégia, será desenvolvido um processo simples. Cada etapa será exemplificada no processo que está sendo desenvolvido.

8.1 Especificação

Esta etapa é realizada através de reuniões com o cliente que solicitou o processo de Workflow. Todos os dados relevantes do processo devem ser documentados, conforme indicado no documento Documentação mínima do Projeto (veja a seção 6.1). Ao final desta etapa, o modo como o processo irá funcionar deve estar bem definido.

No nosso exemplo, será desenvolvido um processo que gerencie “Solicitações”, melhor descrito como segue:

1. Um usuário do processo abre uma nova solicitação;
2. A solicitação é encaminhada para um usuário distribuidor (pessoa que gerencia as solicitações), para ser avaliada;
3. O distribuidor avalia a solicitação e toma uma decisão: aprovar ou rejeitar;
4. Caso a solicitação seja aprovada, ela vai para execução de um técnico que descreve o procedimento que foi feito para solucionar o problema. Ao final da execução, o solicitante recebe um e-mail passando informações sobre sua solicitação;
5. Caso a solicitação seja rejeitada, o solicitante recebe um e-mail informando sobre sua solicitação.

8.2 Projeto

Nesta etapa os dados obtidos na etapa de especificação são mapeados para a realidade do Workflow. E, é possível definir alguns dados sobre o processo, por exemplo: quantos perfis ele terá, o número de atividades, o relacionamento entre elas, etc.

No nosso exemplo podemos perceber claramente a existência de três perfis:

1. **Solicitante**: grupo de pessoas que podem fazer solicitações;
2. **Distribuidor**: grupo de pessoas que avaliam as solicitações;
3. **Técnico**: grupo de pessoas que executam as solicitações aprovadas.

As atividades são quatro:

1. **Compor Solicitação**: o perfil "solicitante" faz uma solicitação;
2. **Avaliar**: o perfil "distribuidor" avalia a solicitação;
3. **Executar**: o perfil "tecnico" executa as solicitações aprovadas;
4. **Informar Resultado**: o perfil "solicitante" vê as informações referentes à sua solicitação.

Possíveis caminhos de execução:

1. Compor Solicitação → Avaliar (Aprovada) → Executar → Informar Resultado;
2. Compor Solicitação → Avaliar (Rejeitada) → Informar Resultado.

8.2.1 Criar o Fluxo

Um dos passos mais importantes no desenvolvimento de um processo de Workflow é definir o fluxo do processo. O fluxo consiste no relacionamento das atividades através de transições. É importante também, quando retratar o fluxo, informar os perfis relacionados a cada atividade.

O resultado da criação do fluxo é um dos mais importantes documentos para o processo de Workflow e que por si só é capaz de demonstrar o funcionamento de todo o sistema.

No nosso exemplo, foi criado o fluxo da Figura 8.1

8.2.2 Criar o Processo

Uma vez definido o fluxo do processo, é possível mapeá-lo para a engine do Workflow que estamos utilizando.

Na interface de Administração de Processos, preenchemos:

```
Nome do processo: Solicitações
Descrição: Processo de gerenciamento de solicitações
```

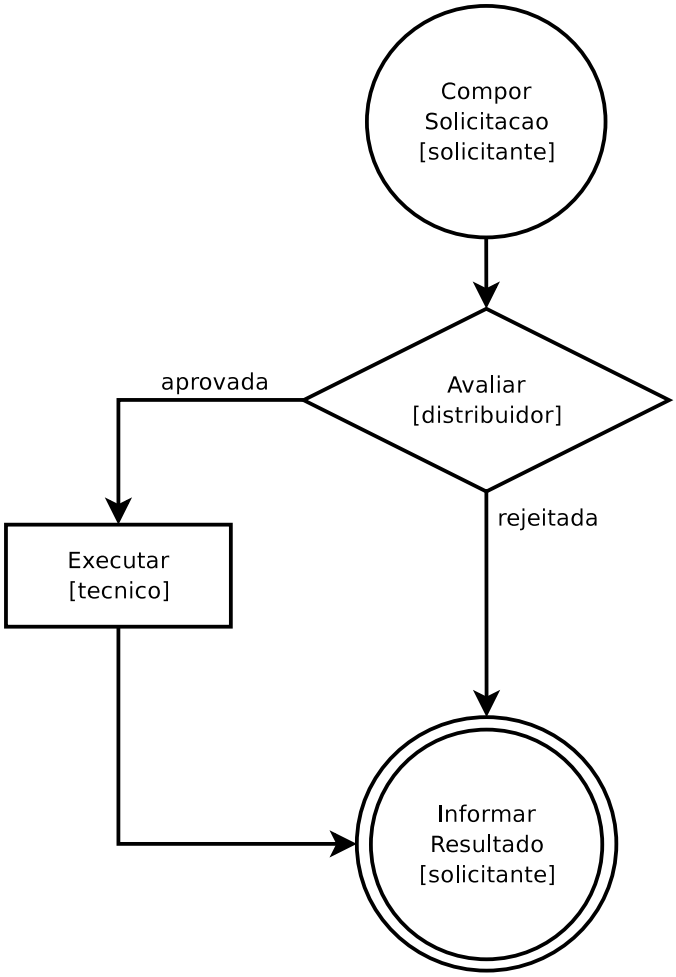


Figura 8.1: Fluxo do processo do tutorial.

Para os outros campos, utilizam-se os valores padrões.

Clicamos no botão “criar”.

8.2.3 Criar as Atividades, Transições e Perfis

Na interface de administração, selecione a opção “Atividades” do processo.

Quando o processo é criado, duas atividades são criadas com ele: a atividade do tipo start e de nome “start” e a atividade do tipo end e de nome “end”.

Selecionamos a atividade “start” e, mudamos/preenchemos os seguintes campos:

```
Nome: Compor Solicitação
Descrição: Atividade para o solicitante compor uma solicitação
Interativa: marcar opção
Roteamento Automático: marcar opção
Adicionar perfil:
  Nome: solicitante
  Descrição: autorizados a compor solicitações
```

Clicamos em “salvar”.

Selecionamos a atividade “end” e, mudamos/preenchemos os seguintes campos:

```
Nome: Informar Resultado
Descrição: Atividade para informar o solicitante sobre sua solicitação
Interativa: marcar opção
Roteamento Automático: marcar opção
Usar perfis existentes:
  Selecionar "solicitante"
```

Clicamos em “salvar”.

Clicamos em “novo”.

Preenchemos os seguintes campos:

```
Nome: Executar
Descrição: Atividade para o técnico executar uma solicitação
Tipo: activity
Interativa: marcar opção
Roteamento Automático: marcar opção
Adicionar Transições:
  Adicionar transições para: "Informar Resultado"
Adicionar perfil:
  Nome: tecnico
  Descrição: autorizados a executar solicitações
```

Clicamos em “salvar”.

Clicamos em “novo”.

Preenchemos os seguintes campos:

```
Nome: Avaliar
Descrição: Atividade para o distribuidor avaliar uma solicitação
Tipo: switch
Interativa: marcar opção
Roteamento Automático: marcar opção
Adicionar Transições:
  Adicionar transições de: "Compor Solicitação"
  Adicionar transições para: "Executar" e "Informar Resultado"
Adicionar perfil:
  Nome: distribuidor
  Descrição: autorizados a avaliar solicitações
```

Clicamos em “salvar”.

8.2.4 Mapear os Perfis

Clicamos em “Perfis”.

Na área “Mapear usuários/grupos a perfis”, selecionamos um dos perfis (no lado direito) e adicionamos os usuários que pertencem a este perfil. Depois, clicamos em “Mapear”.

Para finalizar, basta repetir este mesmo passo para os outros dois perfis.

8.3 Implementação

Nesta etapa é feita a codificação das atividades que foram criadas na etapa anterior. Será utilizada a arquitetura MVC. Consulte a seção 7 para saber mais detalhes de como a mesma está implementada no Workflow do Expresso.

Será utilizado o editor de códigos PHP que já vem embutido no Workflow. Veja a seção 4.4 para maiores detalhes de uso do editor.

8.3.1 Codificar as Atividades

Acesse a interface de administração, e selecione o link “Código” do processo que está sendo desenvolvido. Será aberta a interface de edição de código e na aba “Atividades” estará a lista das atividades componentes do processo. Para cada uma delas insira o código PHP listado mais abaixo.

Nome da Atividade: Avaliar
Nome do Arquivo: Avaliar.php

```
<?php
$application = new AvaliarController(new AvaliarModel($env), $env);
$application->run($_REQUEST['action']);
?>
```

Nome da Atividade: Compor Solicitação
Nome do Arquivo: Compor_Solicitao.php

```
<?php
$application = new ComporSolicitacaoController(new ComporSolicitacaoModel(
    $env), $env);
$application->run($_REQUEST['action']);
?>
```

Nome da Atividade: Executar
Nome do Arquivo: Executar.php

```
<?php
$application = new ExecutarController(new ExecutarModel($env), $env);
$application->run($_REQUEST['action']);
?>
```

Nome da Atividade: Informar Resultado
Nome do Arquivo: Informar_Resultado.php

```
<?php
```

```
$application = new InformarResultadoController(new InformarResultadoModel(
    $env), $env);
$application->run($_REQUEST['action']);
?>
```

8.3.2 Codificar os Templates

Ainda na interface de edição de código, clique na aba “templates”. Para cada template insira o código abaixo. Pode-se copiar e colar o código.

Nome da Atividade: Avaliar
Nome do Arquivo: templates/Avaliar.tpl

```
{include file="info_solicitacao.tpl"}

<br />
<input type="submit" name="action" value="Aprovar" />
<input type="submit" name="action" value="Rejeitar" />
```

Nome da Atividade: Compor Solicitação
Nome do Arquivo: templates/Compor_Solicitao.tpl

```
<table>
  <tr>
    <td><label>Título</label></td>
    <td><input type="text" name="titulo" value="{$titulo}" /></td>
  </tr>
  <tr>
    <td><label>Descrição</label></td>
    <td><textarea name="descricao">{$descricao}</textarea></td>
  </tr>
</table>

<input type="submit" name="action" value="Enviar" />
```

Nome da Atividade: Executar
Nome do Arquivo: templates/Executar.tpl

```
{include file="info_solicitacao.tpl"}
```

```
<br />

<strong>Procedimento Executado:</strong><br/>
<textarea name="procedimento" cols="50" rows="7"></textarea><br />
<input type="submit" name="action" value="Finalizar" />
```

Nome da Atividade: Informar Resultado
Nome do Arquivo: templates/Informar_Resultado.tpl

```
{include file="info_solicitacao.tpl"}
<br />
<table>
  <tr>
    <td><strong>Mensagem</strong></td>
    <td>{$mensagem|nl2br}</td>
  <tr>
</table>

<input type="submit" name="action" value="Encerrar" />
```

Por padrão, o Workflow cria um arquivo tpl para cada atividade interativa do processo. O arquivo info_solicitacao.tpl não corresponde a uma atividade, mas sim a um trecho de código que será incluído em outros arquivos tpl. Por isso, não estará na lista de arquivos já existentes, e deverá ser criado da seguinte maneira:

1. Clique no botão “Novo template”;
2. Escolha a opção “em_branco.tpl”;
3. Informe o nome do arquivo (info_solicitacao.tpl) e clique em OK;
4. Inclua o código abaixo.

Nome do Arquivo: templates/info_solicitacao.tpl

```
<table>
  <tr>
    <td><strong>Solicitante</strong></td>
    <td>{$solicitante_desc}</td>
  </tr>
  <tr>
    <td><strong>Data</strong></td>
    <td>{$data}</td>
```

```
  </tr>
  <tr>
    <td><strong>Título</strong></td>
    <td>{$titulo}</td>
  </tr>
  <tr>
    <td><strong>Descrição</strong></td>
    <td>{$descricao|nl2br}</td>
  </tr>
</table>
```

8.3.3 Codificar os Includes

Os próximos arquivos devem ser incluídos de forma semelhante como foi feito para o arquivo `info_solicitacao.tpl` anteriormente, só que desta vez será usada a aba "Includes", da interface de código. Para cada arquivo a ser incluído, execute:

1. Clique no botão "Novo Include";
2. Escolha a opção "`em_branco.php`";
3. Informe o nome do arquivo e clique em OK;
4. Inclua o código correspondente.

Nome do Arquivo: `class.avaliar.controller.inc.php`

```
<?php
class AvaliarController extends Controller
{
  function __default ()
  {
    $this->model->defaultAction();
    $this->loadViewVars();
    $this->showForm($this->AVALIAR);
  }

  function aprovar()
  {
    $this->model->aprovarAction();
  }
```

```
  function rejeitar()
  {
    $this->model->rejeitarAction();
  }
}
?>
```

Nome do Arquivo: `class.avaliar.model.inc.php`

```
<?php
class AvaliarModel extends Model
{
  function defaultAction()
  {
    $this->updateAttributes();
    $this->addViewVar('titulo', $this->_titulo);
    $this->addViewVar('descricao', $this->_descricao);
    $this->addViewVar('data', $this->_data);
    $this->addViewVar('solicitante_desc', $this->_solicitante_desc);

    return true;
  }

  function aprovarAction()
  {
    $this->instance->setNextActivity('Executar');
    $this->commitInstance();

    return true;
  }

  function rejeitarAction()
  {
    $this->updateAttributes();
    $this->instance->setNextActivity('Informar Resultado');
    $this->instance->setNextUser($this->_solicitante); /* devolve a
       instância para o solicitante */
    $this->_mensagem = "Sua solicitação foi rejeitada";
    $this->updateInstance();
    $this->commitInstance();

    return true;
```

```
  }
}
?>
```

Nome do Arquivo: `class.compor.solicitacao.controller.inc.php`

```
<?php
class ComporSolicitacaoController extends Controller
{
  function __default ()
  {
    $this->model->defaultAction();
    $this->loadViewVars();
    $this->showForm($this->COMPOR_SOLICITACAO);
  }

  function enviar()
  {
    $this->model->enviarAction();
    $this->loadViewVars();
  }
}
?>
```

Nome do Arquivo: `class.compor.solicitacao.model.inc.php`

```
<?php
class ComporSolicitacaoModel extends Model
{
  function defaultAction()
  {
    return true;
  }

  function inputValidate($form)
  {
    $msgerro = Array();

    /* título não pode ser vazio */
    if (isset($form['titulo']) && !empty($form['titulo']))
      $this->_titulo = $form['titulo'];
    else
```

```
      $msgerro[] = 'É necessário fornecer um título';

    if (isset($form['descricao']) && !empty($form['descricao']))
      $this->_descricao = $form['descricao'];
    else
      $msgerro[] = 'É necessário fornecer uma descrição';

    return $msgerro;
  }

  function enviarAction()
  {
    /* se não houve erros */
    if (count($this->activity->error = $this->inputValidate($this->request))
        == 0)
    {
      $this->_solicitante = $this->getWfProperty('wf_user_id');
      $this->_solicitante_desc = $this->getWfProperty('wf_user_cnname');
      $this->_data = date('d/m/Y H\hi');
      $this->updateInstance();
      $this->commitInstance();

      return true;
    }
    else
    {
      $this->addViewVar('titulo', $this->_titulo);
      $this->addViewVar('descricao', $this->_descricao);
      return false;
    }
  }
}
?>
```

Nome do Arquivo: `class.controller.inc.php`

```
<?php
class Controller extends BaseController
{
  var $COMPOR_SOLICITACAO = 'Compor_Solicitao.tpl';
  var $AVALIAR = 'Avaliar.tpl';
  var $EXECUTAR = 'Executar.tpl';
```

```
  var $INFORMAR_RESULTADO = 'Informar_Resultado.tpl';

  function Controller(&$model , &$env)
  {
    $this->super(&$model , &$env);
  }

  function run($action)
  {
    $this->dispatch($action);
  }
}
?>
```

Nome do Arquivo: `class.executar.controller.inc.php`

```
?php
class ExecutarController extends Controller
{
  function __default ()
  {
    $this->model->defaultAction();
    $this->loadViewVars();
    $this->showForm($this->EXECUTAR);
  }

  function finalizar()
  {
    $this->model->finalizarAction();
  }
}
?>
```

Nome do Arquivo: `class.executar.model.inc.php`

```
<?php
class ExecutarModel extends Model
{
  function defaultAction()
  {
    $this->updateAttributes();
    $this->addViewVar('titulo', $this->_titulo);
```

```
    $this->addViewVar('descricao', $this->_descricao);
    $this->addViewVar('data', $this->_data);
    $this->addViewVar('solicitante_desc', $this->_solicitante_desc);

    return true;
  }

  function finalizarAction()
  {
    $this->updateAttributes();
    $this->_mensagem = "Sua solicitação foi atendida pelo técnico "
      . $this->getWfProperty('wf_user_cnname')
      . ".\n<strong>Procedimento executado:</strong>\n" . $this->request['
        procedimento'];
    $this->updateInstance();
    $this->instance->setNextUser($this->_solicitante); /* devolve a
       instância para o solicitante */
    $this->commitInstance();

    return true;
  }
}
?>
```

Nome do Arquivo: `class.informar.resultado.controller.inc.php`

```
<?php
class InformarResultadoController extends Controller
{
  function __default ()
  {
    $this->model->defaultAction();
    $this->loadViewVars();
    $this->showForm($this->INFORMAR_RESULTADO);
  }

  function encerrar()
  {
    $this->model->encerrarAction();
  }
}
?>
```

Nome do Arquivo: `class.informar.resultado.model.inc.php`

```
<?php
class InformarResultadoModel extends Model
{
  function defaultAction()
  {
    $this->updateAttributes();
    $this->addViewVar('titulo', $this->_titulo);
    $this->addViewVar('descricao', $this->_descricao);
    $this->addViewVar('data', $this->_data);
    $this->addViewVar('solicitante_desc', $this->_solicitante_desc);
    $this->addViewVar('mensagem', $this->_mensagem);

    return true;
  }

  function encerrarAction()
  {
    $this->commitInstance();
    return true;
  }
}
?>
```

Nome do Arquivo: `class.model.inc.php`

```
<?php
class Model extends BaseModel
{
  var $_titulo;
  var $_descricao;
  var $_solicitante;
  var $_solicitante_desc;
  var $_data;
  var $_mensagem;

  function Model(&$env)
  {
    $this->super(&$env);
```

```
  }
}
?>
```

Nome do Arquivo: `shared.php`

```
<?php
/* camada de controle do processo */
wf_include('class.controller.inc.php');
wf_include('class.compor.solicitacao.controller.inc.php');
wf_include('class.avaliar.controller.inc.php');
wf_include('class.executar.controller.inc.php');
wf_include('class.informar.resultado.controller.inc.php');

/* camada da lógica de negócios */
wf_include('class.model.inc.php');
wf_include('class.compor.solicitacao.model.inc.php');
wf_include('class.avaliar.model.inc.php');
wf_include('class.executar.model.inc.php');
wf_include('class.informar.resultado.model.inc.php');
?>
```

8.4 Finalização

Agora só falta ativar o processo. Para isto, acesse a interface de Administração de Processos e clique em “Ativar”.

Feito isso, o processo estará pronto para uso. Para executá-lo siga os passos:

1. Abra a interface de usuário (clique sobre o ícone do módulo Workflow);
2. Clique na aba “Processo”;
3. Passe o mouse sobre o ícone do processo para mostrar o menu de atividades.

Obs: o processo estará disponível para quem estiver mapeado aos perfis.

Se quiser colocar um ícone personalizado para o processo, faça:

1. Acessar a interface de código;
2. Clicar a aba “Resources”;
3. Fazer o upload de uma imagem de 32×32 pixels com o nome de “`icon.png`”.

9 AMBIENTE DE DESENVOLVIMENTO

O ambiente de desenvolvimento criado pela CELEPAR pode ser visto na Figura 9.1.

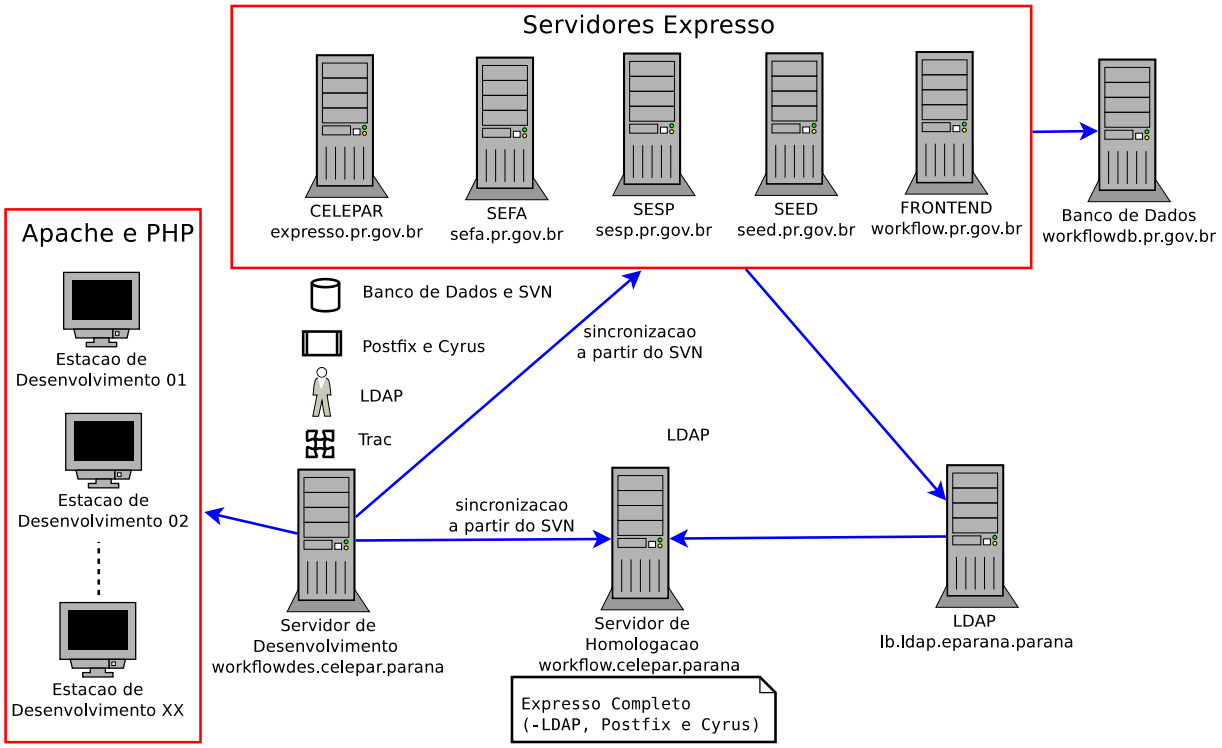


Figura 9.1: Diagrama do ambiente de desenvolvimento.

O Workflow conta com três servidores, são eles:

- Desenvolvimento (`workflowdes.celepar.parana`): utilizado para fornecer os serviços utilizados pelo ExpressoLivre/Workflow das estações de desenvolvimento, além de abrigar os repositórios SVN e o gerenciado de projetos Trac;
- Homologação (`workflow.celepar.parana`): utilizado para a homologação do processo desenvolvido. Este é um ambiente mais próximo do de Produção. Com exceção do LDAP, Postfix e Cyrus, esta máquina provê todos os serviços que utiliza;
- Produção (`workflow.pr.gov.br` e `workflowdb.pr.gov.br`): fornece os serviços de banco de dados e FrontEnd Web para todas as instalações do ExpressoLivre/Workflow vinculadas ao Governo Estadual.

9.1 Conhecendo o Ambiente

O desenvolvimento de processos de Workflow é feito utilizando duas máquinas, são elas:

1. a estação do desenvolvedor;
2. o servidor de desenvolvimento.

A Figura 9.2 mostra a organização do processo na estação do desenvolvedor e como a estação interage com o servidor de desenvolvimento. O desenvolvedor possui em sua estação o Apache,

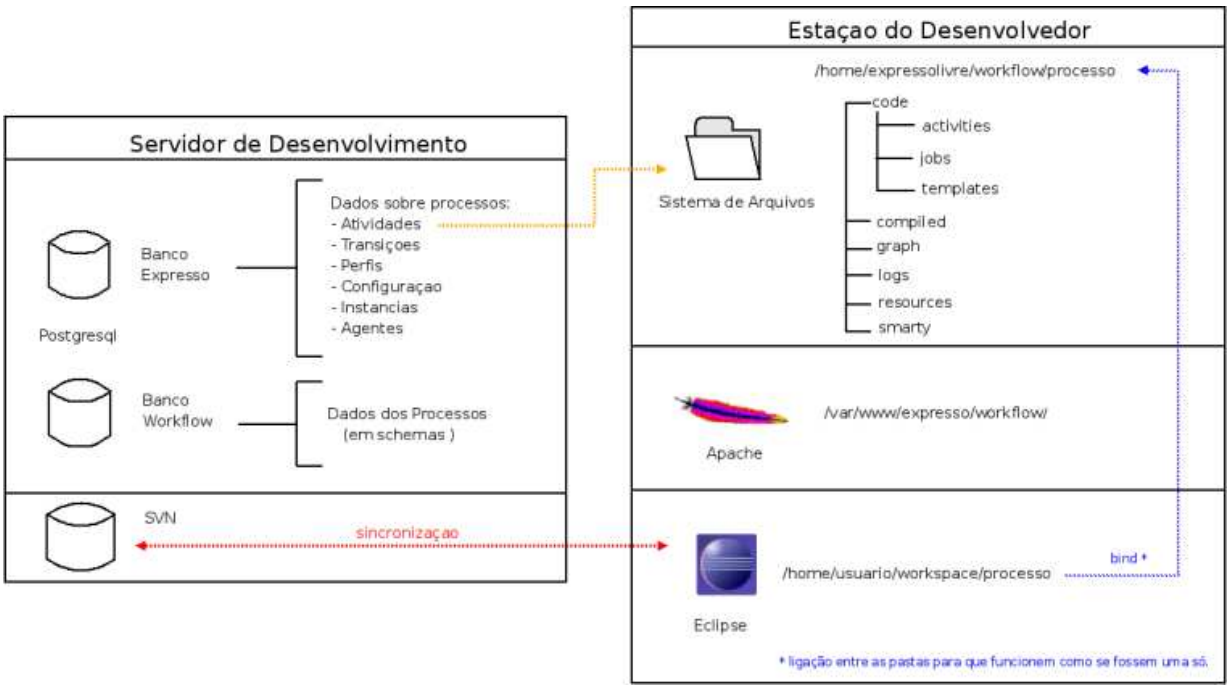


Figura 9.2: Diagrama da interação entre a estação local e o servidor de desenvolvimento.

o código do ExpressoLivre/Workflow, o código do processo que está sendo desenvolvido, e uma IDE, no caso o Eclipse, que possibilita editar o código PHP e sincronizá-lo com um repositório SVN.

O servidor de desenvolvimento, por sua vez, possui os serviços de correio (Postfix e Cyrus), banco de dados (PostgreSQL), LDAP e servidor SVN.

O desenvolvimento é feito desta maneira pois permite que mais de um desenvolvedor trabalhe no mesmo projeto sem riscos de perda da integridade. Pois, o fluxo do processo é armazenado em banco, já o código do processo é armazenado em arquivos locais (vinculados ao SVN). Trabalhando desta forma, mudanças no fluxo do processo se refletem automaticamente para todos

os desenvolvedores (já que todos utilizam o banco de dados do servidor de desenvolvimento) e, mudanças nos dados dos arquivos (php, js, css, imagens, etc.) são feitas localmente, e ao serem comitadas para o SVN, passam a estar disponíveis para os demais desenvolvedores.

9.2 Instalação

O pré-requisito para instalar o ambiente de desenvolvimento é ter uma estação com sistema Operacional Linux, preferencialmente Debian ou Ubuntu.

1. **Atualizar a lista de pacotes de software**:
   - Como root execute o comando:
     `apt-get update`
2. **Executar o script de instalação do ExpressoLivre/Workflow**
   - Baixe o script de instalação[7]
   - Entrar em um shell de comando
   - Mudar para o superusuário (root)
   - Ir para o diretório onde foi baixado o script e executar os comando:
     `chmod 744 workflowInstall.sh`
     `./workflowInstall.sh`
3. **Obter uma conta de Desenvolvedor**:
   - Entre em contato com a administração do workflow e solicite que seja feito o registro do novo desenvolvedor
   - Como root, edite o arquivo `/var/www/expresso/header.inc.php`
   - Mude o valor do atributo `db_user` para o nome da conta de desenvolvimento;
   - Mude o valor do atributo `db_pass` para a senha da conta.
   - Salve o arquivo.
4. **Instalar o Eclipse**:
   - Como root execute o comando:
     `apt-get install eclipse`
5. **Configurar os Plugins do Eclipse**:
   - Abrir o menu Help/Software Updates
   - Adicionar as seguintes localidades:

[7] `https://wfsvn.celepar.parana/Administracao/trunk/scripts/workflowInstall.sh`

```
http://phpeclipse.sourceforge.net/update/stable/1.2.x
http://subclipse.tigris.org/update_1.4.x
```

6. **Periodicamente atualize o Expresso**
   - Como root, execute o comando:
     `svn update /var/www/expresso`

Vale salientar que a instalação do Eclipse (e seus plugins) é opcional. Pode ser utilizado qualquer editor PHP.

9.3 Juntar-se ao Desenvolvimento de um Processo

Antes de começar a desenvolver um processo já existente, é necessário contactar um administrador Workflow para que este possa dar as devidas permissões:

1. Permitir acesso à interface de administração de processos;
2. Permitir edição do referido processo;
3. Permitir acesso ao repositório SVN do processo.

Uma vez que isso tenha sido realizado, proceda os seguintes passos:

- Fazer o download do script de sincronizacao de processos:[8];
- Como root, executar o script:
  `chmod +x dsvn2wf.sh`
  `./dsvn2wf.sh [login_do_usuario]`

Siga os passos indicados pelas telas do script.

9.4 Iniciar o Desenvolvimento de um Novo Processo

Caso esteja sendo iniciado um novo processo, é necessário contactar um administrador do Workflow para que ele:

1. Crie um grupo de desenvolvedores do processo;
2. Crie um novo esquema no banco de dados do servidor de desenvolvimento (caso o processo utilize banco de dados);

[8] `https://wfsvn.celepar.parana/Administracao/trunk/scripts/dsvn2wf.sh`

3. Crie o repositório SVN para versionamento do processo.

A única informação necessária para que isto seja feito, é o nome do processo. No terceiro item, será criado um processo básico, com o código mínimo, para o desenvolvedor começar a desenvolver o seu processo seguindo a arquitetura MVC.

Uma vez que o administrador do Workflow tenha executado estas etapas, basta seguir os passos da seção 9.3.

9.5 *Deploy* de Processos

O *deploy* de processos permite que estes estejam disponíveis em outros ambientes. Os ambientes disponíveis são: (1) ambiente de homologação; (2) ambientes de produção (depende de quais ambientes o seu processo participa).

Basicamente, existem três ações diferentes para o *deploy*:

- **Implantação de um Processo**: utilizada quando um processo não existe no ambiente para o qual se quer fazer o *deploy*. Normalmente, também utiliza o *deploy* de comandos SQL (para a criação de tabelas);
- **Atualização de um Processo já Implantado**: utilizada quando o processo já existe no ambiente destino e, as modificações estão restritas aos arquivos que compõem o processo;
- **Execução de Comandos SQL**: utilizado para criar, alterar ou fazer carga de dados no banco de dados do processo.

Todas estas ações devem ser solicitadas através do processo de Workflow "Transferência de Processos" (atividade: "Solicitar Transferência") do servidor de desenvolvimento[9]. A interface deste processo pode ser vista na Figura 9.3.

9.5.1 Implantação de um Processo

Para solicitar a implantação de um processo (ou seja, que ainda não existe no ambiente destino), é necessário anexar o arquivo XML do processo (gerado pelo módulo Workflow) e, se necessário, os comandos SQL que criam as tabelas (e relações) utilizadas pelo processo. Após

[9] `http://workflowdes.celepar.parana`

Figura 9.3: Interface do processo “Transferência de Processos”.

a importação (feita por um administrador do Workflow), o solicitante deve fazer o mapeamento dos perfis e ativar o processo.

Para obter o arquivo XML, o solicitante deve acessar a interface de administração de processos e, para o processo que será implantado, selecionar a opção “Salvar” (ver Figura 9.4).

Figura 9.4: Ações administrativas de um processo.

9.5.2 Atualização de um Processo já Implantado

No caso da atualização, supõe-se que o processo já foi previamente implantado no ambiente destino. Esta ação, trata somente modificações nos arquivos (alteração, inclusão ou exclusão).

Mudanças de fluxo, inclusão ou exclusão de atividades, mudanças nos perfis, mudanças nas propriedades do processo ou das atividades não são atualizadas através desta ação. Sendo assim, estas devem ser feitas pelo solicitante diretamente no ambiente de destino.

Caso seja incluída uma atividade nova e seja necessário atualizar algum ambiente, o procedimento deverá ser este:

1. Incluir a atividade manualmente no ambiente de destino (associação de perfis, fluxo desta atividade, etc.). Não é necessário colocar o código da atividade (PHP, TPL, ect.);
2. Fazer uma solicitação de transferência dos arquivos do processo (assim, o código da nova atividade será disponibilizado no ambiente destino).

Apenas para reforçar a idéia, esta ação apenas sincroniza os arquivos de código (dentro do diretório `code`) e de recursos (dentro do diretório `resources`).

Os arquivos que serão transferidos para o ambiente destino são extraídos diretamente do SVN. O processo é atualizado por inteiro, não havendo a possibilidade de atualizar, por exemplo, apenas um arquivo.

Ao fazer uma solicitação deste tipo, também é possível selecionar uma versão específica do processo no SVN. Isto possibilita, por exemplo, que o desenvolvedor que está refazendo totalmente um processo, possa criar um branch da versão implantada para fazer pequenas modificações e, solicitar sincronizações deste branch enquanto a alteração maior não está pronta. O padrão ao se solicitar uma transferência é a utilização do ramo `trunk` (versão mais recente).

9.5.3 Execução de Comandos SQL

Esta ação engloba os casos onde são necessárias apenas alterações no banco de dados.

Estas alterações podem ser do tipo: DML (`SELECT`, `INSERT`, `UPDATE` e `DELETE`) e DDL (`CREATE` e `DROP`).